ALGORITHMES DE COMPRESSION D'IMAGES ET CODES DE CONTOURS

Ehoud Ahronovitz

Année 1985 N° d'ordre 173

THESE

présentée devant l'Université de
SAINT-ETIENNE

pour obtenir le titre de
DOCTEUR DE 3ème CYCLE
INFORMATIQUE

par
Ehoud AHRONOVITZ

ALGORITHMES DE COMPRESSION D'IMAGES ET CODES DE CONTOURS

préparée à l'Ecole des Mines de Saint-Etienne

soutenue le 30 Septembre 1985 devant la commission d'examen:

Président	M.	J.C. SIMON
Examinateurs	MM.	J.P. GOURE
		M. HABIB
	Mle.	P. JOLY
	MM.	M. JOURLIN
		B. PEROCHE
	Mme.	F. ROMEO

Année 1985 N° d'ordre 173

THESE

présentée devant l'Université de
SAINT-ETIENNE

pour obtenir le titre de
DOCTEUR DE 3ème CYCLE
INFORMATIQUE

par
Ehoud AHRONOVITZ

ALGORITHMES DE COMPRESSION D'IMAGES ET CODES DE CONTOURS

préparée à l'Ecole des Mines de Saint-Etienne

soutenue le 30 Septembre 1985 devant la commission d'examen:

Président	M.	J.C. SIMON
Examinateurs	MM.	J.P. GOURE
		M. HABIB
	Mle.	P. JOLY
	MM.	M. JOURLIN
		B. PEROCHE
	Mme.	F. ROMEO

Monsieur Jean-Claude SIMON a bien voulu me faire l'honneur de présider ce jury et je désire lui exprimer ma profonde reconnaissance pour l'intérêt qu'il a porté à notre travail.

La réalisation et les essais matériels ont été faits en collaboration avec le CNET-PARIS A et je remercie en particulier Pascale JOLY et Françoise ROMEO qui nous ont consacré beaucoup de leur temps et ont accepté la méthode proposée. Leurs critiques très constructives nous ont été fort utiles. Je suis heureux de les voir participer à ce jury.

Que Jean-Pierre GOURE et Michel JOURLIN du laboratoire "Traitement du Signal" à l'UER Sciences de Saint-Etienne, et Bernard PEROCHE de l'équipe "Communication visuelle" à l'Ecole des Mines de Saint-Etienne, veuillent bien trouver ici l'expression de mes remerciements pour leur soutien, remarques et encouragements.

Comment remercier Michel HABIB? Le dynamisme, la méthode avec lesquels il a dirigé ce travail dépassent le cadre induit par quelques mots; je souhaiterais simplement continuer à travailler avec lui, de la même façon, le plus longtemps possible.

La naissance et les débuts de vie du produit développé sont dus à Marc BERTIER et aussi à Jean-Claude MOISSINAC. J'espère que les évolutions actuelles leur donnent satisfaction.

Enfin, je remercie:

- Fleury VELAY, Louis DARLES et André LOUBET qui ont réalisé le tirage de cet ouvrage, en acceptant avec beaucoup de sympathie les retards successifs,

- les membres du département informatique de l'Ecole des Mines, et tous mes amis, qui ont su être indulgents lorsque je n'étais pas enclin à l'être,

- et surtout ma femme, qui a subi ce travail directement en temps qu'informaticienne, et m'a supporté pendant sa réalisation et sa rédaction.

Résurgence stéphanoise de troff/nroff, GROFF, formateur de texte (J.J. GIRARDOT & SYNC) a réalisé toute la mise en page de cette thèse.

TABLE DES MATIERES

CHAPITRE 4

UNE APPLICATION
COMPRESSION DE DOCUMENTS TECHNIQUES

INTRODUCTION

Le traitement d'images sur ordinateur nécessite une transformation analogique / numérique.

Les principes de ces transformations sont exposés par SIMON [SIM1, SIM2] ou PAVLIDIS [PAVL] par exemple. Il est important de garder présent à l'esprit, que la numérisation d'une image analogique et à plus forte raison, sa binarisation, qu'elles soient logicielles ou matérielles, sont des codages parfois dégradés de l'original, et que c'est cette image numérisée (i.e. une matrice de pixels) que nous devons considérer comme la donnée de départ.

Le volume important de cette donnée, fait que le problème de son codage sous forme condensée a été très tôt étudié (CAPON, GILBERT &MOORE, 1959), qu'il s'agisse d'en optimiser le stockage ou la transmission. En effet, une image d'un document de taille A4 numérisée avec un facteur de résolution de 8 points par mm représente environ 4 millions de points. Qui plus est, l'obtention d'une qualité accrue sur les matériels de visualisation ou reproduction, rend nécessaire une plus grande finesse de la saisie, et les volumes des images ont alors tendance à s'accroître.

Le travail décrit ci-dessous, a eu comme motivation initiale un problème concret qui nous a été posé par le CNET-PARIS A, dans le cadre du projet SARDE[1], de télématisation de la documentation technique des télécommunications.

Les problèmes que nous traitons concernent la réduction du volume d'images bicolores, afin de diminuer l'espace nécessaire au stockage des documents, ainsi que la restitution de l'image sous la forme la plus fidèle possible à l'original.

---

1 SARDE est une marque déposée par le CNET

Les documents que nous avons considérés sont des documents de bureau, ou encore des schémas techniques, manuscrits ou non, en noir et blanc.

Nous prévoyons dans une étape ultérieure l'extension à des images multicolores; il nous semble que les principes que nous développons rendent cette adaptation plausible.

Nous n'envisageons pas les problèmes de bases de données et ne parlons pas des outils informatiques nécessaires à leur mise en oeuvre.

Bien que dans le cas de la transmission à distance, se pose le problème de la fiabiblité (code correcteur d'erreurs de transmission), nous ne nous intéresserons pas ici à ce critère. Nous considérons que c'est au système de transmission de données utilisé d'assurer l'intégrité des informations qui lui sont confiées.

Dans le premier chapitre, nous étudions les principales techniques logicielles de compression actuellement connues. Toute méthode de compression doit résoudre les compromis

- taux de compression / temps de calcul
  et
- taux de compression / fidélité de restitution.

Malheureusement, il n'est pas toujours facile d'obtenir des éléments de comparaison, du fait du développement constant des diverses méthodes. Nous avons été contraints de reconstituer quelques algorithmes ou résultats; nous espérons que ces reconstitutions restent fidèles aux idées des auteurs.

Le deuxième chapitre détaille la méthode que nous utilisons pour l'extraction des objets de l'image, ainsi que deux techniques d'affichage sur des écrans haute résolution dotés d'une mémoire d'image ("bitmap").

Le développement technologique d'outils basés sur le balayage type télévision a permis l'émergence de nouveaux algorithmes qui suivent le balayage. CEDERBERG a le premier utilisé ces techniques de balayage sur des problèmes de compression d'images [CED1, CED2]. Ses travaux nous ont permis de proposer une méthode originale d'extraction des contours des objets contenus dans l'image, et le calcul de quelques caractéristiques propres à ces objets, en un seul balayage de l'image.

Disposer de telles techniques linéaires permet de profiter pleinement de tout accroissement de la puissance des machines, puisqu'une multiplication par K de cette puissance engendre dans ces cas une division par K des temps de calcul. Aussi, nous proposons dans la suite, chaque fois que possible, des algorithmes linéaires en fonction du nombre d'éléments à traiter.

Nous avons développé deux techniques d'affichage (restitution): la première, par balayage, représente un peu l'inverse du codage; la deuxième, baptisée "affichage aléatoire", n'est possible que grâce aux simplifications que nous avons apportées au codage, et nous permet de proposer une méthode reposant sur une "algèbre de contours".

Dans le chapitre 3 nous étudions les spécificités et l'optimisation des codes de contours. Nous proposons des éléments de caractérisation des objets manipulés avant la phase de compression.

Bien que nous ne visions pas la compréhension de l'image, problème difficile et central de la reconnaissance des formes [SIM1], nous y contribuons par la détection d'objets ou morceaux d'objets caractéristiques, et l'interface avec des méthodes de reconnaissance de formes nous paraît possible.

Le chapitre 4 décrit une application réalisée en collaboration avec le CNET (division OGE/TDT). Nous avons validé notre méthode en réalisant un logiciel prototype sur une machine SM90 (développée au CNET), sous le système Unix[1] (version SMX).

Nous constituons une bibliothèque de contours pour chaque image et cherchons des similitudes entre objets. Là aussi, nous sommes encore loin de la reconnaissance des formes. Néanmoins, nous profitons des particularités de notre code de contours pour définir des "points caractéristiques" des formes. Nous nous en servons comme base de comparaison entre les éléments déjà connus (bibliothèque), et les nouveaux candidats.

Dans la phase de compression finale, nous nous heurtons à un autre compromis, propre à toute méthode dite "symbolique":

---

1 Unix est une marque déposée de AT&T.

optimisation des références à la bibliothèque /
optimisation de la localisation des objets.

La meilleure optimisation des références à la bibliothèque est obtenue en regroupant tous les objets semblables contenus dans l'image. Il faut alors traiter entièrement toute l'image, avant d'obtenir le codage définitif. Par contre, l'utilisation en télécopie, où codeur et décodeur sont enchaînés, implique l'optimisation de la localisation, afin de ne pas bloquer la transmission inutilement, jusqu'à la fin du traitement de l'image.

Nous avons réalisé d'abord la première, dans le but d'étudier les performances de l'algorithme de comparaison des contours. La deuxième est en cours de développement.

Les essais ont été faits sur 14 images propres au CNET, et sur 5 des 8 images-test du CCITT[1]. En taux de compression, nous obtenons sur ces dernières, par rapport à la méthode de JOHNSEN & al [JOHN], 3 meilleurs résultats un résultat équivalent, et un moins bon. De plus, sur les 3 images qui nous manquent actuellement, il nous semble qu'au moins deux d'entre elles nous sont favorables.

Ces premiers résultats nous permettent de penser que la méthode élaborée possède des bases intéressantes. Nous continuons actuellement à y apporter des améliorations, et proposons des développements futurs dans plusieurs directions (cf. conclusion). La méthode actuelle fera l'objet d'une communication aux journées SM90[2].

Enfin, ce travail nous a permis de soulever de nombreux problèmes théoriques, que ce soient:

- les manipulations d'images par codes de contours,

- ou certains problèmes liés aux polyominos,

qu'il nous semble intéressant d'approfondir par la suite.

---

1 Comité Consultatif International des Téléphones et Télégraphes

2 journées INRIA ADI CNET, Versailles, Décembre 1985

CHAPITRE 1

LES METHODES DE COMPRESSION

Nous passons ici en revue quelques méthodes de compression connues actuellement, en étudiant les principes qui les régissent et leur adaptation/adaptabilité à diverses utilisations. Les comparaisons se font souvent par rapport à la norme CCITT actuelle (voir codage READ). Les documents ("images") traités sont de taille A4 et nous envisageons la question des documents de taille supérieure et de meilleure qualité (résolution plus fine).

1. PRINCIPES ET TECHNIQUES DE BASE.

On distingue souvent deux classes de méthodes de compression: les méthodes exactes et les méthodes approchées. Elles peuvent être applicables à n'importe quel type d'image, ou plus adaptées aux images bicolores. Toutefois, le même principe peut, très souvent, permettre la réalisation de variantes de l'un et de l'autre type. Il est donc plus intéressant de classer les méthodes suivant les principes qu'elles mettent en oeuvre.

Quel que soit le principe d'une méthode de compression, il exploite l'idée que l'on peut coder plus efficacement les messages à transmettre (ou à stocker) dans la mesure où toutes les valeurs que peuvent prendre ces messages ne sont pas équiprobables.

Ainsi le code de Huffman consiste-t-il, connaissant a priori la fréquence d'apparition de tous les messages, à imposer un code court aux messages fréquents et inversement [HUFF]. Dans le cas des images, l'information à transmettre est la couleur de tous les points de la grille d'échantillonnage:

$$(x_1, y_1, \text{coul}(x_1, y_1)), \ldots, (x_k, y_k, \text{coul}(x_k, y_k))$$

où $\text{coul}(x_i, y_i)$=noir ou blanc,

et k=nombre de pixels de l'image.

Evidemment, on peut imposer un ordre de parcours commun au codeur et au décodeur, c'est-à-dire définir (x_{i+1} , y_{i+1}) en fonction de (x_i , y_i). Il suffit alors de transmettre:

coul(x_1,y_1), ... coul(x_k,y_k)

soit 1 bit par point de l'image.

C'est ce code, que l'on appellera binaire, qui servira de référence dans la suite pour définir le taux de compression obtenu avec une autre méthode de codage.

Bien qu'il y ait une très grande disproportion entre le nombre de pixels noirs et le nombre de pixels blancs, le codage de Huffman, appliqué à une image bicolore, ne procure aucun gain car on ne peut avoir de codes plus courts que un bit!

Si la disproportion entre le nombre de pixels noirs et blancs est élevée, on peut transmettre les seules coordonnées des points noirs (ou plus généralement de la couleur la moins représentée)

(x_1,y_1), (x_2,y_2), ... (x_n,y_n)

Si xi et yi sont codés sur p bits chacun, cette méthode réalise une compression -par rapport au code binaire- à condition que:

(nbre pixels noirs) / (nbre total pixels) < 2.p

Une autre technique appelée Codage par Blocs [KUNT] consiste à découper l'image en rectangles juxtaposés, de taille k par l. On choisit k et l de manière à ce que:

- la probabilité d'avoir un rectangle k par l entièrement blanc soit élevée
- la taille du bloc soit maximale.

Les blocs blancs sont alors codés sur 1 bit, et les autres sur (k.l)+1 bits.

2. *DEUXIEME APPROCHE.*

Dans les images bicolores représentant les documents de bureaux, les pixels noirs et blancs ne sont pas dispersés aléatoirement mais sont en général regroupés. Il est alors plus efficace de coder la limite des zones de couleurs différentes. Deux classes de méthodes utilisent cela:

2.1. LE CODAGE DE FREEMAN.

Le codage de Freeman [FREE] ("chain link code") du contour des taches noires: à partir d'un pixel noir qui a au moins un voisin blanc, on suit le contour, par exemple dans le sens des aiguilles d'une montre, et à chaque fois que l'on passe d'un point x au suivant, on code la direction qui permet de passer de l'un à l'autre (8 valeurs possibles, soit 3 bits).

Figure 1.1
codage de Freeman

```
5 6 7          1 2 3 4 5 6
4 x 0        1 . . . . . .
3 2 1        2 . . X X . .
             3 . . X X X .
             4 . X X . X .
             5 . . . . . .
```

sera codé: (2,3),0,1,2,5,3,4,7,6

2.2. LE CODAGE PAR PLAGES.

Le codage par plages ("run length code"), lié au mode de balayage de l'écran (balayage ligne par ligne ou "raster scan"), consiste à coder la longueur des plages monochromes qui se succèdent sur les lignes du balayage. Dans l'exemple ci-dessus, un codage par plages donnera:

fdl,2,2,fdl,2,3,fdl,1,2,1,1,fdl,fdl

où fdl code la fin de ligne.

Remarques:

1. Du fait même de leur principe commun, les deux types de méthodes ont approximativement les mêmes cas favorables et défavorables, selon la valeur du rapport:

(nombre de pixels noirs) / (nombre de pixels de contour)

2. Les images présentant de longs morceaux de contours horizontaux sont favorables au code par plages, sinon les deux codes donnent le même nombre de messages (égal au nombre de pixels de contour), chaque message est sur 3 bits pour le code de Freeman, et sur p bits pour le code par plages, si 2^p est la largeur de l'image).

3. Si l'on traite des pages de textes dactylographiés, l'image bicolore sera formée d'un grand nombre de petites taches noires disjointes d'épaisseur à peu près constante. Ceci conduit à utiliser un code de Huffman pour la longueur des plages (code fac-simile standard CCITT).

4. Les directions en 2 pixels consécutifs du contour, sont fortement corrélées; les performances du codage de Freeman peuvent être sensiblement améliorées,

- en introduisant un facteur de répétition dans le code,

- ou en codant la différence entre la direction en un pixel et la direction au pixel précédent, "code différentiel", avec un code très déséquilibré:

Direction	Code
0	0
+1	10
-1	110
+2	1110
etc...	

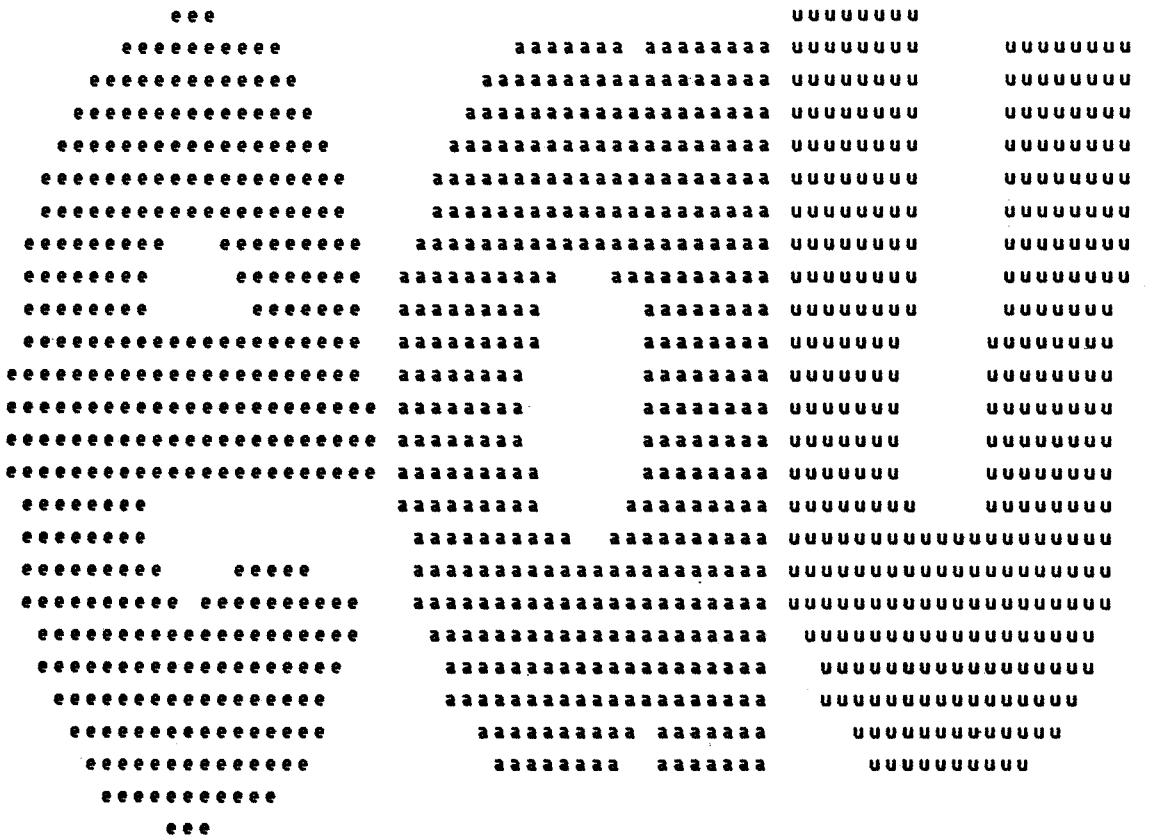
L'exemple précédent a pour code différentiel:

((2,3),0),+1,+1,+3,-2,+1,+3,-1

La deuxième solution semble ici largement préférable à la première (nos tentatives ont confirmé nos lectures!).

Une solution mixte est possible: décaler les codes à partir de -2 pour introduire un code spécial pour k zéros consécutifs (k>=6). -cf. figure suivante-

Figure 1.2
Comparaison sommaire des tailles



	"e"	"a"	"u"
Code binaire	598	552	480
Code plages	310	350	390
Chain link	324	309	312
Ch-lk differ.	181	161	146

Question:

Est-il possible de combiner l'exploitation de la corrélation de la couleur des points dans toutes les directions, d'une manière compatible avec le balayage ligne à ligne?

A nouveau deux types de méthodes se présentent qui répondent à cette contrainte et constituent une adaptation d'un codage par plages pour la première, et une adaptation d'un codage par contours pour la deuxième.

Le code READ ("Relative Element Address Designate coding") [PROC] est le meilleur des codages par plages bidimensionnels actuellement connu (extension du standard CCITT).

Le principe en est le suivant:

- la première de k lignes consécutives est normalement codée par plages (code fac-simile Huffman modifié);

- les plages des k-1 lignes qui suivent sont codées,

  . soit normalement par leurs longueurs,

  . soit par la différence entre leurs longueurs et celles des plages correspondantes dans la ligne précédente.

3. CODAGE PAR CONTOURS COMPATIBLE AVEC LE BALAYAGE LIGNE.

Le codage par contour compatible avec le balayage ligne à ligne [CED1, CED2] a introduit un code par contour dont la construction peut être effectuée au fur et à mesure du balayage ligne à ligne de l'écran, le RC-code (pour "Raster scan Chain link code").

3.1. PRINCIPE.

Le balayage ligne à ligne introduit un ordre entre 2 pixels de contour voisins. Cet ordre permet de poser les définitions suivantes:

- point de naissance: point de contour sans prédécesseur;

- point de mort: point de contour sans successeur;

- paroi: liste des points de contour d'un point de naissance à un point de mort.

Le codage des chaînages dans le RC-code n'utilise que 4 mots qui suffisent à représenter tous les liens entre un pixel de contour x et son successeur:

```
  x 0
3 2 1
```

Le codage minimal obtenu à l'aide du RC-code sur une image bicolore, est appelé RC-code compact.

Il est formé:

- de la liste des points de naissance des parois des composantes connexes, dans l'ordre du balayage;

- puis de la liste des chaînages rencontrés dans l'ordre du balayage (les chaînages des diverses parois sont donc mélangés; par contre les 2 listes sont indépendantes!). Dans le cas de la figure 1.1 on obtient:

points de naissance : (2,3), (3,4)

points de mort: (4,3), (4,5)

chaînages : 2 0 1 3 3 1 2 0

dans l'ordre: (2,3) 2 0 1 3 (3,4) 3 1 2 0 (4,3) (4,5)

parois: 2 3 0 , 0 1 2 , 3 , 1

Remarque:

Le gain en taux de compression est minime par rapport au chain-link classique (2/3 au mieux) et bien inférieur à ce que permet le chain-link différentiel. Cederberg propose d'ailleurs une version différentielle du RC-code.

L'intérêt du principe réside avant tout dans le mode de construction (et de décodage) au fil du balayage.

3.2. CONSTRUCTION DU RC-CODE COMPACT.

Au fur et à mesure du balayage ligne à ligne, le voisinage 3x3 d'un point noir permet d'affirmer

- qu'il s'agit d'un point de naissance qui doit donc être stocké en queue de la liste correspondante;
- qu'il s'agit d'un point de chaînage dont la direction doit donc être stockée en fin de liste des chaînages.

3.3. DECODAGE DU RC-CODE COMPACT.

Etant donné le RC-code compact d'une image binaire -i.e. une liste de points de naissance et un liste de chaînages-, le décodage peut lui aussi être réalisé en suivant le balayage ligne à ligne; il suffit de disposer de l'indication pour chaque point de la ligne précédente, du nombre de parois passant par ce point.

3.4. RC-CODE PAR ENTITES.

Cederberg propose une version plus sophistiquée du RC-code. Il l'appelle "entity RC-code".

Le stockage des points de naissance et des chaînages rencontrés au fur et à mesure du balayage, dans une structure de données plus complexe que les 2 listes précédentes, permet, dès que l'on atteint le point de mort qui ferme un contour, de disposer de la structure qui relie ses différentes parois.

On peut alors réaliser facilement toute opération nécessitant une description complète du contour.

4. PRINCIPE DES METHODES SYMBOLIQUES.

Les codes par plages et par contour supposent et exploitent le regroupement de pixels en taches. Rien de plus!

Si on fait des suppositions plus fortes sur la structure de l'image binaire à comprimer, on pourra exploiter cette information a priori.

Exemple 1:

Si l'on traite des images formées de lignes brisées (schémas techniques par exemple), on peut coder l'image sous forme des bi-points à relier. On parle alors de vectorisation. Plusieurs techniques sont présentées dans la littérature, mais la prise en compte et la restitution de l'épaisseur des traits sont toujours délicates.

Exemple 2:

Dans le cas des documents dactylographiés, Asher et Nagy [ASHE] proposent d'exploiter l'apparition à de multiples endroits dans une page, de petites composantes connexes presque identiques: les symboles.

Au fur et à mesure du codage, les nouveaux symboles rencontrés sont stockés dans une structure de données appelée bibliothèque.

Tout symbole extrait est comparé aux symboles déjà dans la bibliothèque pour décider s'il est un nouveau symbole ou s'il est déjà apparu précédemment. Dans ce dernier cas, il est simplement codé par une référence en bibliothèque.

Sur cette idée, plusieurs méthodes ont été proposées depuis. Le principe commun est présenté par l'algorithme suivant:

```
BIB_SYMB: liste de couples (numéro, description d'un symbole)

BIB_SYMB := vide
tant que non FIN_DE_DOC faire
        répéter
              P := POINTSUIVANT      (* balayage ligne à ligne *)
       jusqu'à FIN_DE_DOC ou P=NOIR
       si P=NOIR alors
              S := SYMBOLE_EXTRAIT        (* à partir de P *)
              CHERCHER S DANS BIB-SYMB
              si TROUVE alors
                     CODE := NUMERO  (* de S dans BIB_SYMB *)
              sinon
                     INTRODUIRE S DANS BIB_SYMB
                     CODE := S               (*  cf. remarque  *)
              finsi
              ENVOYER CODE
       finsi
fintant
CODE := FIN_DE_DOC
ENVOYER CODE
```

Remarque:

La description explicite du symbole est transmise à sa première occurrence. La bibliothèque de tous les symboles n'est jamais transmise globalement, mais seulement au fur et à mesure du codage; ainsi le décodeur reconstruira-t-il une bibliothèque qui évoluera au cours du décodage de la même manière que lors du codage.

Parmi les variantes bâties sur ce principe, certaines sont dites mixtes ou hybrides: du fait que l'extraction d'un symbole à partir d'un point P peut échouer dès que l'objet dépasse la taille maximale prévue pour un symbole, la zone noire -appelée "résidu"- à laquelle P appartient, est alors l'objet d'un autre type de codage.

5. QUELQUES METHODES SYMBOLIQUES ET/OU MIXTES.

5.1. C.S.M. COMBINED SYMBOL MATCHING.

Cette méthode mixte (Pratt et al. [PROC]), utilise deux techniques de compression déjà sommairement décrites plus haut:

- la reconnaissance de symboles,

- le codage par plages bidimensionnel (READ).

5.1.1. Principe.

Nous décrivons ci-après la trame de l'algorithme (reconstitution!); mais il faut noter que le codage d'un symbole nécessite l'accès anticipé aux n lignes suivantes de

l'image, où n est la hauteur maximale d'un symbole.

```
BIB_SYMB := vide
tant que non FIN_DE_DOC faire
        LIRE_1_LIGNE (LIG[1..NP])
        (* extraire et coder les symboles *)
        I := 1
        tant que I<=NP faire
                si LIG[I]=BLANC alors
                        I := I+1
                sinon    (* c'est donc un point clé *)
                        EXTRAIRE_SYMB (REUSSI, S)
                        si REUSSI alors
                                (* coder S ou son numero *)
                                EFFACER S
                        sinon
                                répéter
                                        I := I+1
                                jusqu'à
                                (LIG[I]=BLANC ou I>NP)
                        finsi
                finsi
        fintant

        (* coder le résidu de la ligne *)
        I := 1
        tant que I<=NP faire
                CODE_READ (LIG[1..NP])
        fintant
fintant
```

5.1.2. Extraction des symboles.

A partir d'un point clé, i.e. un point noir précédé d'un point blanc sur la ligne de balayage, le premier travail est de tenter l'extraction ou "mise en boite" ("blocking") d'un symbole. Pratt et al. n'indiquent pas dans l'article la méthode utilisée, mais il semble qu'il s'agit d'un suivi du contour (à la fin duquel on peut disposer de la hauteur et de la largeur du symbole), puis un remplissage de ce contour avec les pixels de l'image. On peut supposer que cette étape est un des points sensibles en temps d'exécution. Cependant, l'opération peut être cablée ou encore parallélisée.

5.1.3. Recherche d'un symbole dans la bibliothèque.

Si la "mise en boite" réussit, -i.e. une petite composante connexe a été isolée-, on doit chercher le symbole ainsi isolé dans la bibliothèque des symboles déjà apparus; et, suivant le cas,

- transmettre l'identification du symbole dans la bibliothèque,

- ou l'ajouter à la bibliothèque, transmettre sa description, et l'effacer de l'image.

Comme il est a priori coûteux de comparer précisement deux symboles ("Matching"), il faut éviter la comparaison systématique du symbole candidat à tous ceux de la bibliothèque. Pour cela sont associés aux symboles, quatre descripteurs: la hauteur, la largeur, le périmètre et la surface.

On calcule alors la distance de Manhattan ("city block distance" i.e. la somme de la valeur absolue de la différence sur chaque descripteur) entre le candidat et tous les éléments de la bibliothèque (que nous appellerons "prototypes"). On réalise alors la comparaison précise du candidat aux prototypes, dans l'ordre croissant des distances au candidat.

On arrête la comparaison dès qu'elle est positive (symbole reconnu) ou que la distance dépasse un certain seuil.

5.1.4. Comparaison d'un symbole candidat et d'un prototype.

La comparaison entre deux symboles en code binaire est réalisée en trois étapes:

- le OU-EXCLUSIF bit-à-bit donne les points où les deux symboles diffèrent (image d'erreur).

- les points d'erreur recoivent comme valeur v, le nombre de points d'erreur dans le carré 3x3 dont ils sont le centre ($1 <= v <= 9$).

- la somme de ces valeurs pour tous les points est effectuée; et ce calcul est répété pour neuf positions relatives des deux symboles (translations horizontales et verticales de -1, 0, +1), et la valeur minimale est comparée à un seuil fonction -non linéaire tabulée- du nombre de pixels noirs pour décider de l'égalité ou non de deux symboles.

Bien que l'extraction de caractéristiques bien discriminantes réduise le nombre de comparaisons, cette opération est probablement encore un point sensible en temps d'exécution.

5.1.5. Gestion de la bibliothèque des symboles.

Les nouveaux symboles sont rangés dans une bibliothèque de taille fixe. Chaque entrée occupée contient:

- le symbole lui-même (sous quelle forme?),

- les quatre descripteurs associés,

- un compteur de référence (cf ci-dessous).

Un mécanisme de pointage (lequel?) permet, dans le cas où la bibliothèque est pleine, d'expulser le caractère "le moins utilisé", pour en mettre un nouveau à sa place. Bien que cela ne soit pas explicitement dit, il semble que le critère utilisé soit celui de "moins récemment utilisé", de manière à ce que le nouveau prototype introduit, ne soit pas ensuite le premier rejeté.

5.1.6. Transmission d'un nouveau symbole.

La transmission d'un prototype commence par sa largeur et sa hauteur. Ensuite, deux méthodes sont proposées, suivant le compromis complexité/taux-de-compression désiré.

- code binaire le long du balayage ligne à ligne,

- code de Huffman des longueurs de plages le long d'un balayage oblique (gain 30 pour cent).

Le codage du résidu fait l'objet du codage par plages bi-dimensionnel standard CCITT (READ).

5.2. METHODE "HYBRIDE" CNET-OGE.

Dans le cadre du projet SARDE d'archivage de la documentation technique des Télécommunications, une nouvelle méthode tenant compte des spécificités de cette documentation a été élaborée par Roméo et Joly [ROM1] [ROM2].

La première originalité de cette méthode est d'enchaîner non pas deux mais trois techniques de codage (dans l'ordre):

1. la vectorisation des graphismes,

2. la reconnaissance de symboles,

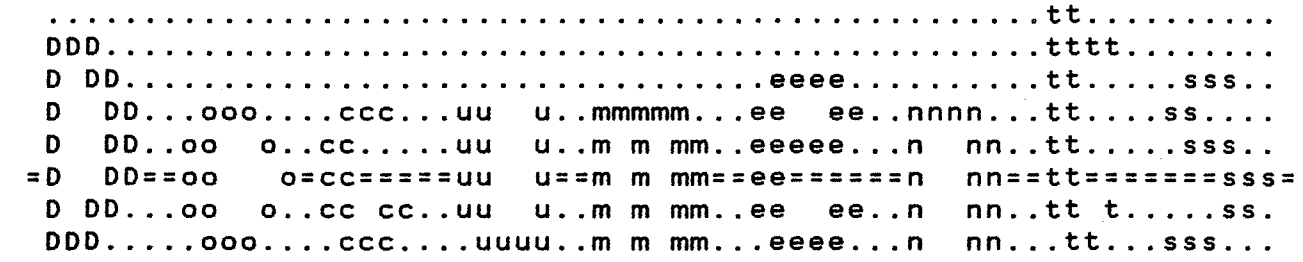
3. le codage READ du résidu,
   d'où le terme de "méthode hybride".

L'introduction d'une technique de vectorisation est tout à fait motivée a priori par la proportion importante de plans techniques dans l'ensemble des documents issus de la documentation technique des télécommunications.

Une autre particularité importante de la méthode réside dans le choix d'optimiser le codage de la localisation et non pas le numéro du symbole comme font Johnsen et al. [JOHN] (décrite dans le paragraphe suivant). Ceci est obtenu en codant les caractères, dans l'ordre lexicographique. Pour cela, la méthode commence par placer une ligne sonde qui "embroche" tous les caractères d'une ligne

de texte (à la condition qu'elle soit à peu près horizontale -voir figure -.

Figure 1.3
Influence de la ligne sonde.



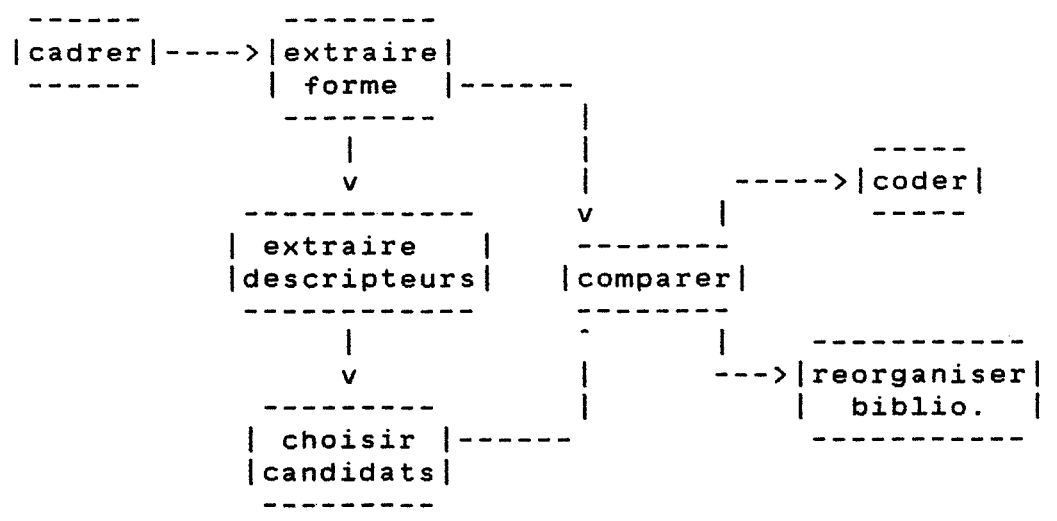
5.3. UNE METHODE SYMBOLIQUE /BELL LABS - O. JOHNSEN ET AL./

Contrairement aux deux précédentes cette méthode n'est pas mixte mais purement symbolique. Elle consiste à isoler deux types de formes: symboles et non-symboles, et à leur appliquer le même traitement (d'où son nom);

Un symbole est une forme connexe entièrement encadrée dans une fenêtre de taille prédéfinie (ce sera souvent un caractère). Un non-symbole est un morceau d'une composante connexe obtenu par fenêtrage.

Chaque forme est comparée à des éléments de bibliothèque. Lorsqu'il y a correspondance, la position de la forme et l'identification en bibliothèque sont codées; sinon, la nouvelle forme est ajoutée à la bibliothèque.

Le schéma de principe est:



On peut remarquer les particularités suivantes:

- Les non-symboles sont codés comme les symboles. Ainsi il y a uniformisation du traitement et ceci est assez original. Il n'y a pas de "résidu" et les "grosses taches noires" sont représentées par des morceaux (non-symboles) correspondants à la taille de la fenêtre dans laquelle on cherche à encadrer les formes.

- Il y a une seule bibliothèque pour les symboles et les non-symboles.

- chaque ligne de balayage non vide est réorganisée avant d'être transmise.

- la bibliothèque est mise à jour après la transmission de chaque ligne.

5.3.1. Fenêtrage et extraction des formes.

La fenêtre est de taille carrée prédéfinie W . Le premier point noir est cadré au centre-haut d'une fenêtre de taille 2W*W. La recherche du contour est classique (8-connexe); les points noirs isolés sont éliminés. Un symbole est une forme entièrement cadrée dans une fenêtre. Un non-symbole a au moins un point noir connexe à l'extérieur. Toute forme cadrée est transmise pour la suite de son traitement, et enlevée (mise à blanc) de la page pour la suite du balayage.

5.3.2. Extraction des caractéristiques.

Elle se fait sur 4 critères: largeur et longueur de la forme, puis nombre de segments blancs horizontaux et verticaux intérieurs à la forme. Un segment blanc est considéré comme intérieur à la forme lorsqu'il permet de rejoindre 2 points noirs.

5.3.3. Sélection des candidats dans la bibliothèque.

Elle s'opère sur les 4 caractéristiques précédentes. Les candidats possibles sont pris en vue de la comparaison dans l'ordre de la bibliothèque, qui correspond au nombre de fois où un candidat a été effectivement retenu. Les auteurs parlent de deux passes, la deuxième "moins stricte" que la première, mais il est difficile de connaître le(s) "seuils de sélection": à partir de quel moment un élément de bibliothèque devient candidat?

5.3.4. Comparaison.

On obtient une image d'erreurs par un "ou exclusif" entre le candidat et la forme. Cette opération est faite 9 fois avec des déplacements horizontaux et verticaux de un point à la fois.

Comme dans CSM l'image d'erreurs est pondérée pour chaque point par le nombre de voisins; les erreurs isolées (0 voisins-erreurs) sont abandonnées. Par contre, la décision de rejet d'un candidat tient compte de l'origine des erreurs et il y a rejet lorsqu'une "tache d'erreurs" provient exclusivement de l'une des deux figures comparées.

5.3.5. Codage.

Chaque ligne de balayage n'est transmise qu'après l'analyse complète de toute la ligne. Le but est de rassembler les formes reconnues identiques pour raccourcir le codage.

Une ligne (de balayage) ne contenant aucune forme est codée sur un seul bit; à la fin de chaque ligne contenant des formes un code de fin de ligne est transmis permettant ainsi de séparer les lignes.

L'analyse de toute une ligne revient à:

1. isoler les formes de la ligne

2. trier les formes reconnues (dans la bibliothèque) par ordre croissant du numéro d'identification ; la fréquence d'apparition de chaque forme dans la ligne sera prise en compte après la transmission de la ligne, pour la mise à jour de la bibliothèque.

3. ajouter les nouvelles formes en bout de ligne dans l'ordre séquentiel.

Toute ligne codée se présente sous la forme:

ligne non vide	formes identifiées en bibliothèque	nouvelles formes	fin de ligne

Chaque forme identifiée est décrite par sa position horizontale absolue dans la ligne, la référence (numéro) de l'élément de bibliothèque, et le deplacement vertical (provenant de la comparaison) par rapport à cet élément.

Une nouvelle forme est décrite par son code bi-dimensionnel standard CCITT, la taille de ce code, et la position horizontale absolue dans la ligne.

5.3.6. Mise à jour de la bibliothèque.

Il faut noter que la bibliothèque d'une page est construite pendant l'analyse de cette page mais elle n'est jamais transmise ni conservée dans le code définitif. Seuls les contenus des lignes sous la forme qu'on vient de voir sont transmis et permettent de construire à nouveau la bibliothèque lors du décodage.

Pour ce faire il faut que codeur et décodeur aient les mêmes règles de création/mise à jour de la bibliothèque. Ainsi un élément supprimé de la bibliothèque (parce qu'elle est pleine et l'élément est peu fréquent) peut fort bien y revenir comme nouvel élément plus tard. La physionomie de la bibliothèque change à chaque ligne mais doit rester stable tant que la ligne n'a pas été entièrement codée.

5.3.7. Résultats.

Qualité des documents:

Quelques distorsions non significatives (jugement des auteurs).

Compression:

Taux de compression obtenu sur les documents officiels du CCITT:

de 15.3 jusqu'à 63.1 (CCITT: de 7.4 à 47.5).

Sur les documents "non officiels" (numérisation sur 2128 x 1728 points):

de 16.3 à 82.3 (CCITT: de 7.1 à 45.4).

Sur cette dernière version C.S.M obtient de 9 à 63.1 .

Les éléments gourmands dans le codage sont la description des nouveaux éléments (60% du total de la page) et le codage de la position horizontale (20%).

Les résultats semblent dépendre de la qualité de la binarisation de façon plus importante que le code CCITT. Il est intéressant de noter que les auteurs classent dans l'ordre les facteurs d'amélioration suivants:

1. le fenêtrage et codage des non-symboles,

2. l'identification des formes dans la bibliothèque,

et seulement ensuite:

3. l'efficacité obtenue par classement de la bibliothèque et "quelques autres facteurs"... Ils suggèrent un matériel à "logique rapide" pour la recherche des contours qui est à la fois complexe et consommatrice de temps, et une parallélisation possible pour les comparaisons avec la bibliothèque, considérées comme non difficiles mais consommatrices de temps.

6. PROBLEMES LIES AUX GRANDS FORMATS.

Question:

Qu'est ce qui caractérise un grand format?

- Une taille supérieure au format de l'organe de saisie?

- ou bien une taille supérieure à ce que l'on peut charger en mémoire vive?

Nous abordons ici ce problème sous le deuxième aspect. Dans ce cas, le problème est très lié:

- au mode d'accès à l'image imposé par la méthode mise en oeuvre, à savoir, le morceau d'image qu'il est nécessaire de stocker en mémoire pour procéder au traitement; ce morceau peut aller d'un pixel, jusqu'à l'image entière;

- aux structures de données construites par les programmes.

6.1. MODE D'ACCES A L'IMAGE.

Le simple codage par plages demande que l'on accède aux pixels un à un, et dans l'ordre du balayage ligne à ligne. La limitation sur le format disparait complètement.

La vectorisation classique utilisée au CNET demande d'accéder aux pixels en deux passes de balayage horizontale puis verticale, ce qui n'est pratique qu'à condition de disposer de l'image entière en mémoire vive.

Le codage par contour (Freeman classique) nécessite l'accès aléatoire aux pixels.

Le codage des contours selon la technique de Cederberg, suit strictement le balayage ligne à ligne. A chaque instant on doit disposer en plus de la ligne courante, de la ligne précédente et de la ligne suivante.

C.S.M. et les autres méthodes symboliques, suivent grossièrement le balayage télévision, à une importante nuance près: l'accès direct est nécessaire par anticipation sur une cinquantaine de lignes d'image devant la ligne de balayage.

6.2. STRUCTURES DE DONNEES CONSTRUITES PAR LES PROGRAMMES.

La taille de la mémoire vive nécessaire selon le format du document traité, est aussi liée aux structures de données que le programme peut être amené à construire pendant le codage. Ce problème est réglé de manière très simple dans C.S.M. et dans la méthode de Johnsen et al., puisque:

- la taille d'un symbole est limitée (de l'ordre de 50x50)

- la taille de la bibliothèque aussi (512 ou 1024 entrées)

Par contre ce problème n'est pas trivial pour la technique de Cederberg. En effet, le nombre de parois des composantes connexes qui coupent la ligne de balayage à un instant, peut être grand. De plus, chacune de ces parois peut s'étendre du début à la fin de l'image, et avoir donc un code de contour volumineux (par exemple dans le cas des documents quadrillés ou comportant des cartouches).

Une solution extrêmement simple existe: elle consiste, lorsque la structure de donnée atteint la taille maximale, à simuler la présence d'une ligne blanche. Ceci revient donc simplement à découper un document de structure trop volumineuse exactement au meilleur endroit.

6.3. DECOUPAGE DES GRANDS FORMATS.

Si la possibilité d'accès direct doit effectivement être conservée, le traitement des "grands formats" peut être réalisé par découpe en formats inférieurs dans la limite disponible. On peut envisager:

- soit que la saisie réalise le découpage et que l'étape de compression ne s'en soucie pas.

- soit le contraire.

La première solution ne doit pas poser de problème (simplement un document n'est pas une référence disque, mais un descripteur contenant des références à des morceaux comprimés ou non), mais elle a l'inconvénient de faire apparaître au niveau de la saisie une limitation due au traitement et qui peut de plus changer ultérieurement.

La deuxième solution consiste à enchaîner autant d'étapes élémentaires de compression de morceaux tenant en mémoire que nécessaire, en conservant la même bibliothèque de symboles.

6.4. RECOLLAGE.

Il n'y a pas de raison pour qu'apparaissent de graves difficultés à la restitution. Si le codage et le décodage de chaque partie sont fidèles, la juxtaposition des parties décodées le sera aussi:

- si la ligne de partage coupe un symbole, les 2 morceaux constitueront des symboles qui n'apparaîtront qu'une seule fois, et seront donc codés de manière exacte;

- si la ligne de partage coupe le résidu, le codage des 2 parties sera également exact;

- si elle coupe un vecteur, on aura au plus une discontinuité égale au double de l'erreur maximale admise sur chaque vecteur.

En fait le problème de recollage apparaît si l'on veut construire un code qui représente le document dans sa globalité et non par morceaux (par exemple pour retrouver l'ordre lexicographique global, ou pour reconstituer des vecteurs ou des symboles à cheval sur 2 sous-documents).

La nécessité d'une telle spécification ne nous apparaît pas; cependant, elle est automatiquement atteinte par la solution que nous décrivons pour tenter d'échapper aux limites dues à la taille mémoire.

6.5. UNE PROPOSITION.

L'espace mémoire disponible peut introduire des limites sur le format d'un document à traiter.

Pour repousser cette limite, nous pensons que l'on doit envisager une méthode compatible avec le balayage ligne à ligne.

Nous envisageons un schéma de compression complètement compatible avec le balayage télé, pour construire le code des contours des composantes connexes rencontrées, du type RC-code (cf. plus haut [CED1], [CED2]).

Cette démarche nous permet aussi de répondre facilement à une modification du facteur de résolution. Les problèmes posés par une telle modification sont du même ordre que ceux des grands formats, et nous devons aussi en tenir compte car l'amélioration de la qualité des documents implique une résolution plus fine (12 points par mm).

CHAPITRE 2

LES CODES DE CONTOURS DETECTION ET AFFICHAGE

Nous décrivons la technique d'extraction des objets, composantes connexes formées d'une liste de contours, réalisée en un seul balayage de l'image. La mesure des attributs comme par exemple la largeur la hauteur ou la surface des objets est effectuée en même temps. Les structures de données nécessaires sont détaillées. Pour juger du résultat de l'extraction, nous proposons deux techniques d'affichage (décodage), en étudiant les caractéristiques et les algorithmes de chacune d'elles. La première effectue un balayage de bas en haut et de droite à gauche compte tenu des conventions prises dans le chapitre précédent; la deuxième utilise des propriétés simples des graphes permettant de définir leur extérieur et intérieur (cf. Euler [EULE] selon Saint-Lague [SAIN]).

1. INTERET.

L'extraction des contours par balayage permet de considérer une page saisie comme un flot de bits: à aucun moment il n'est nécessaire de posséder l'image entière pour la traiter, et dans notre cas seules deux lignes consécutives sont utiles. Cette indépendance par rapport à l'échelle de saisie permet une conception où le format de la page et le facteur de résolution n'entrent pas en jeu.

La méthode que nous décrivons ici s'inspire du principe de codage des contours au fil du balayage ligne à ligne proposé par Cederberg, présentée dans le chapitre précédent.

Tout l'intérêt de la technique que nous développons, par rapport à celle de Cederberg, réside dans les importantes simplifications qu'elle apporte. En effet, il paraît naturel dans les méthodes de détection de contours, de définir la frontière des objets non pas par la liste des pixels noirs voisins, mais plutôt par la ligne brisée passant entre les pixels de bord de l'objet et du fond.

Cette idée a été exploitée d'abord par Danielson [DANI] puis par nous-mêmes (indépendamment...). Elle permet aussi de lever la difficulté inhérente aux objets d'un seul pixel d'épaisseur dans la méthode de Cederberg.

Nous y contribuons par la mesure de certains attributs au fil du même balayage (de détection des objets), et par un traitement particulier lors de l'affichage permettant des opérations ensemblistes sur les objets (voir affichage "aléatoire").

2. *QUELQUES DEFINITIONS.*

2.1. CONTOUR.

Nous abandonnons la convention fréquement choisie de définir un contour comme **une liste de pixels noirs**, liste dont chaque élément est **voisin de son successeur.**

Nous utilisons une autre convention consistant à définir un contour comme **la liste des frontières communes à un pixel noir et à un pixel blanc**, liste dans laquelle chaque élément horizontal ou vertical est appelé **un lien**, et est connecté à son successeur. Un contour est alors "une ligne brisée se glissant **entre** les pixels" et dont les sommets seront appelés "interpixels".

Cette deuxième convention a la propriété de décrire de la même manière une image et l'image inverse, les objets et le fond jouant des rôles parfaitement symétriques.

Mais **l'intérêt principal de ce choix** pour l'extraction des contours au fil du balayage réside dans le fait qu'il suffit dès lors, pour déterminer le lien partant d'un "interpixel", d'analyser un voisinage 2x2 autour de celui-ci, au lieu du voisinage 3x3 autour d'un pixel.

La conséquence simplificatrice est multiple:

- la mise en place dans un mot mémoire du voisinage à analyser, étape qu'il est simple et très avantageux de faire réaliser dans le matériel d'interface avec le numériseur, est encore simplifiée;

- le traitement programmé de cette donnée ne donne plus lieu qu'à 14 cas au lieu de 254, et tous ces cas sont très faciles à traiter.

- le cas particulier des objets ayant des morceaux d'un pixel d'épaisseur disparaît.

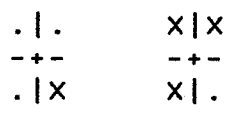
2.2. COMPOSANTE CONNEXE.

Une composante connexe est représentée par un contour (dit extérieur), contenant éventuellement d'autres contours (intérieurs) disjoints. Lorsqu'un contour est inclus dans un contour intérieur, il forme une autre composante connexe.

2.3. POINTS DE NAISSANCE ET DE MORT.

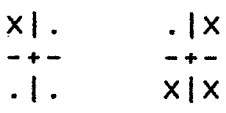
On appelle **point de naissance** un "interpixel" dont le voisinage 2x2 est identique à l'un des deux représentés ci-dessous:

Figure 2.1
Points de naissance



De même un **point de mort** correspond à l'une des configurations suivantes:

Figure 2.2
Points de mort



2.4. PAROI.

Une **paroi** est un tronçon de contour depuis un point de naissance jusqu'à un point de mort.

Les liens qui décrivent une paroi sont soit horizontaux dirigés de gauche à droite, soit verticaux du haut vers le bas; ils peuvent donc être codés sur 1 bit; nous avons choisi la correspondance suivante:

lien horizontal = 0

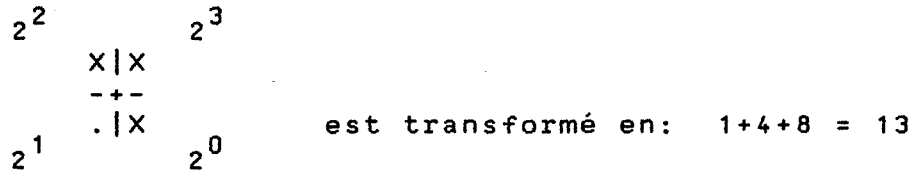
lien vertical = 1

A chaque instant, les parois que coupe la ligne de balayage sont dites **actives**.

3. *PRECODAGE DES "INTERPIXELS" DE CONTOUR.*

Cette première opération consiste simplement à disposer les 4 valeurs binaires représentant le voisinage 2x2 d'un interpixel, dans les bits 0 à 3 d'un mot mémoire (cf. figure suivante).

Figure 2.3
Exemple de précodage d'un interpixel.



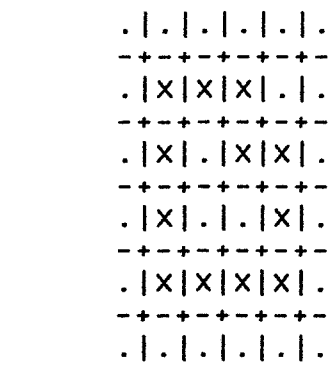
La valeur numérique obtenue doit être transmise uniquement si elle ne correspond pas à un voisinage 2x2 uniformément blanc (0) ou uniformément noir (15).

On obtient ainsi le précodage tel qu'il est illustré dans la figure 2.4

Pour que le code d'une composante permette de la localiser, les coordonnées d'un point de référence sont nécessaires. C'est pourquoi **les précodes correspondant à un point de mort sont suivis des coordonnées de l'interpixel.**

Nous avons choisi les points de mort comme points de référence, et non pas les points de naissance, car cela évite de mémoriser de nombreux couples de coordonnées; en effet, lors de la prise en compte d'un précode de point de mort, on peut déterminer immédiatement si celui-ci ferme un contour et on ne mémorise ses coordonnées que dans ce cas.

Figure 2.4
Exemple de précodage.



sera précodé:

1 3 3 2 9 14 13 7 2 9 6 8 13 6 9 7

3 11 x1 y1 6 8 12 12 12 4 x2 y2

La programmation de cette opération est simple. Toutefois, il est indispensable d'envisager son intégration au prétraitement assuré par le matériel d'interface avec le numériseur.

Le gain à attendre en ce qui concerne le temps de traitement, est très important: il est nécessaire sinon, de calculer tous les interpixels, et de plus, pour les interpixels utiles, le volume des données passe d'1 bit par pixel à 4 bits par interpixel.

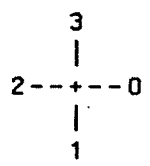
4. EXTRACTION DES COMPOSANTES CONNEXES.

4.1. PRINCIPE DE CONSTRUCTION DU CODE DE CONTOUR.

Ce module reçoit en entrée des codes (appelés désormais **précodes**) qui représentent le voisinage non monochrome d'un interpixel (cf. ci-dessus), et élabore **un code par chaînage du contour** extérieur et éventuellement des contours intérieurs, de chaque composante connexe.

Le code par chaînage est proche de celui de Freeman, mais là encore nous codons une liste d'interpixels où le code de Freeman est formé d'une liste de pixels. La convention adoptée est:

Figure 2.5



L'exemple de la figure 2.4 aura comme code:

nombre de contours: 2

(x2, y2)
2 2 2 2 3 3 3 3 0 0 0 1 0 1 1 1

(x1, y1)
2 2 3 3 0 1 0 1

A chaque fois qu'il reçoit un précode, ce module est capable de mettre à jour le code par chaînage de la paroi idoine grâce à une structure de données interne au module: **l'anneau des parois actives.**

Dans cette structure de données, chaque paroi est mémorisée sous la forme de la liste de liens horizontaux et verticaux qui la code depuis son point de naissance jusqu'au point atteint par la ligne de balayage courante.

La réception du précode d'un point de mort détermine une liaison entre deux parois. Lorsque cette liaison ferme un contour on construit le code de ce contour. Si de plus ce contour est extérieur, il délimite une composante connexe; on peut alors construire le code de contour global de cette composante, et le transmettre en sortie au module suivant.

4.2. ANNEAU DES PAROIS ACTIVES.

Cette structure est une liste simplement chaînée des parois actives, parmi lesquelles on désigne, à tout instant, 3 parois consécutives:

- la Paroi Courante (PC),
- la Paroi Suivante (PS),
- et la paRoi PRécédente (RPR).

Cette liste est initialement vide et on l'appelle "anneau" car le dernier élément est connecté au premier.

Les opérations à effectuer sur cette structure de données sont les suivantes:

- pnaiss(): introduire 2 nouvelles parois avant le pointeur courant; la 2ème des parois insérées devient la nouvelle paroi courante.

- lien_0(): ajouter un lien horizontal en fin de paroi courante. La paroi courante ne change pas.

- lien_1(): ajouter un lien vertical en fin de paroi courante et passer à la paroi suivante.

- pmort(): extraire du chaînage la paroi courante et la suivante, puis désigner comme paroi courante celle qui suivait les 2 parois extraites.

L'opération à effectuer selon le précode reçu, est indiquée sur la figure suivante:

Figure 2.6
action à exécuter pour chaque précode.

précode	action
1	pnaiss
2	lien_1
3	lien_0
4	pmort
5	*
6	lien_1
7	lien_0
8	lien_0
9	lien_1
10	*
11	pmort
12	lien_0
13	lien_1
14	pnaiss

* Le traitement correspondant aux précodes 5 et 10, dépend de la convention choisie pour définir la relation de connexité entre pixels: il s'agit de décider dans ces cas (2 pixels noirs diagonaux), si l'on a une séparation entre deux contours, ou si le contour détecté jusque là continue. La figure suivante indique la ou les opérations à effectuer pour chacune des conventions admissibles.

Figure 2.7
actions dépendant de la convention de connexité.

CONVENTION	précode 5 X|. -+- .|X	précode 10 .|X -+- X|.
-1-		
4-connexité du fond et 8-connexité des objets	lien_1 lien_0	pmort pnaiss
-2-		
6-connexité (1ère diagonale)	pmort pnaiss	pmort pnaiss
-3-		
6-connexité (2ème diagonale)	lien_1 lien_0	lien_1 lien_0
-4-		
8-connexité du fond et 4-connexité des objets	pmort pnaiss	lien_1 lien_0

Nous avons choisi la convention 1 dans toute la suite. Les cas où ce choix modifie les traitements sont signalés.

Description de la structure de l'anneau des parois actives:

```
struct paroi {

    struct paroi
           *suiv,    /* paroi suivante */
           *soeur,   /* paroi de même pt de naissance */
           *conn,    /* paroi de même pt de mort */
           *pap;     /* paroi appariée */

    struct paire
           *ppaire;  /* voir paire de parois actives */

    char   *pll;     /* ptr à la liste de liens */
    int    lgr;      /* lgr en bits de la paroi */
    int    xp;       /* coord courante de l'intersection
                        avec la ligne de balayage */
     int   dg;       /* indicateur droite/gauche */

     };
```

On remarque que de nombreuses informations sont associées à chaque paroi. La description en viendra dans la suite.

4.3. ON TRAITE UN POINT DE MORT.

Comme indiqué plus haut, le traitement d'un point de mort consiste à désactiver, c'est-à-dire à extraire de l'anneau des parois actives, les deux parois qui se rejoignent et se terminent en ce point.

Si ce point de mort ferme un contour, on construit son code global par chaînage.

Dans le cas contraire, le code des parois devenues inactives doit rester accessible bien que devant sortir de l'anneau des parois actives. C'est à cela que servent les pointeurs:

-pconn, qui relie à n'importe quelle paroi, l'autre paroi de même point de mort qu'elle, et
-psoeur, qui relie à n'importe quelle paroi, l'autre paroi de même point de naissance qu'elle.

Ainsi, à l'occurrence du dernier point de mort d'un contour, peut-on suivre toutes les parois de celui-ci pour construire le code de contour global.

Encore faut-il pouvoir discerner si un point de mort ferme ou non un contour. Pour cela, deux solutions apparaissent:

- suivre alternativement les chaînages pconn et psoeur jusqu'à retomber sur le point de départ, ou sur un chaînage NIL;
- attribuer à chaque paroi créée une classe; si un point de mort connecte deux parois de même classe, il ferme un contour, sinon les classes des deux parois sont fusionnées.

Seule la fermeture du contour est en jeu ici. Les chaînages pconn et psoeur sont nécessaires pour construire le code global du contour dans les deux cas, dès que l'on sait que le contour est fermé. Dans le premier cas ces chaînages sont parcourus à chaque point de mort, alors que dans le deuxième ils ne le sont qu'à la fin.

Nous préférons la deuxième solution qui s'avère intéressante non seulement au titre de la fermeture du contour, mais aussi pour l'extraction de quelques caractéristiques des contours.

La classe associée à toute paroi est appelée "paire de parois actives".

4.4. PAIRES DE PAROIS ACTIVES.

En fait une classe d'équivalence contient toujours deux parois actives exactement.

On tient donc à jour, au cours du balayage, l'appariement des parois actives. En un point de naissance les deux parois créées sont appariées. Si un point de mort relie deux parois appariées, il ferme un contour; sinon les deux parois associées aux parois qui se rejoignent et disparaîssent, forment une nouvelle paire (fusion).

Toute paroi active pointe d'une part sur la paire dont elle fait partie, et d'autre part sur l'autre paroi de la paire ("ppaire": pointeur à la paire active et "pap": paroi appariée, dans la structure "paroi").

Des deux parois actives qui forment une paire, l'une est la paroi droite, l'autre la paroi gauche, selon les positions relatives de leurs points d'intersection avec la ligne de balayage.

Au point de naissance, la paroi commençant par un lien 1 est la paroi gauche, celle commençant par 0 est la paroi droite (indicateur droite/gauche dans la structure "paroi").

Les opérations lien_0 et lien_1 ne modifient pas les attributs droite/gauche.

A l'occurrence d'un point mort, l'action à effectuer dépend du cas dans lequel on se trouve:

fusion g-d: si la paroi courante est une paroi gauche et la paroi suivante, une paroi droite, alors les 2 parois sont nécessairement de la même paire; le point de mort ferme un contour, et cette paire disparaît avec les deux parois;

fusion d-g: si PC est une paroi droite et PS une paroi gauche, les attributs gauche/droite des parois appariées ne sont pas modifiés;

fusion g-g: si PC et PS sont toutes les deux des parois gauches, la paroi appariée à PC reste paroi droite, alors que celle appariée à PS devient une paroi gauche;

fusion d-d: inversement, si les deux parois sont droites, la paroi associée à PS reste une paroi gauche, alors que celle associée à PC devient paroi droite.

Cette notion de paire de parois actives s'avère très pratique pour réaliser au fur et à mesure du balayage, le calcul de certains attributs. C'est pourquoi la structure de l'objet "paire" (de parois actives), donnée ci-dessous, fait apparaître des variables traitées dans le paragraphe

"mesure des attributs au fil du balayage".

```
struct paire {

        int sb;                 /* surface déjà balayée */
        int pl;                 /* plage courante */
        int ht;                 /* hauteur */
        int cgauche, cdroite;
                                /* colonne gauche et droite
                                   de la boite englobante
                                   actuellement connue */

        struct contour *premtrou, *derntrou;
                                /* 1er et dernier trous de la
                                   liste associee à cette paire */
        };
```

4.5. REPRESENTATION D'UNE COMPOSANTE CONNEXE.

4.5.1. Structure.

Le passage de la représentation d'une composante connexe sous forme de ses parois à sa représentation sous forme de son (ou ses) contour(s), se fait en autant d'étapes qu'il y a de contours.

Lorsqu'un point de mort ferme un contour, on peut déterminer si ce contour est extérieur ou intérieur.

En effet, puisque le point de mort ferme le contour, il en est le point "le plus en bas à droite", et le point supérieur gauche du voisinage 2x2 auquel il appartient est nécessairement intérieur au contour. Le précode du point de mort indique donc le type du contour:

- précode 4 (et 5) => contour extérieur.

- précode 11 (et 10) => contour intérieur.

A chaque fois qu'un contour est fermé, on en construit le code global par chaînage à partir des codes des parois dans la structure ci-dessous:

```
/* codage global d'un contour comme liste de liens de 0 a 3 */

struct contour {
        int       lgr;     /* longueur en nombre de liens */
        int       x, y;    /* pt de depart du code du contour */
        int       surf;    /* surface englobée par le contour */

        char      *chain; /* ptr à la liste des liens */

        struct    contour
                  *suiv;  /* contour suivant */
        };
```

Si le contour fermé est intérieur, il ne peut pas être transmis tout de suite, mais devra être transmis avec les autres contours de la même composante connexe.

Dans ce cas, la première paroi à gauche (pointée par RPR) fait aussi, nécessairement partie d'un contour de la même composante connexe. Aussi mémorise-t-on la référence au contour que l'on vient de construire dans la description de cette paroi (pointeurs "premtrou" et "derntrou" de la structure "paire"). Une paroi pouvant référencer plusieurs contours, ceux-ci sont chaînés.

Si le contour fermé est un contour extérieur, son code global doit être construit, mais l'on peut maintenant transmettre la description complète de la composante connexe, sous la forme d'un objet dont la structure est présentée ci-dessous:

```
/* représentation d'une composante connexe */

struct ccx {

              /* attributs */
        int     haut, larg,       /* boite englobante */
                nc,               /* nombre de contours */
                sp,               /* somme des perimètres */
                surf;             /* surface noire */

        struct  contour
                *cext;  /* ptr description du contour ext. */
        };
```

Pour construire la liste des contours, il suffit de chaîner entre elles, les listes de contours intérieurs élaborées préalablement.

4.5.2. Gestion des listes de liens.

Logiquement, une paroi est définie par les coordonnées de son point de naissance, et la liste des liens qui la constitue.

Un lien horizontal est représenté par la valeur binaire 0, et un lien vertical par la valeur 1.

Une paroi pouvant aussi bien avoir une longueur petite (un lien) que grande (plusieurs milliers), la liste qui la représente doit nécessairement être gérée dynamiquement.

Il est absolument inefficace de la gérer par chaînage de "structures" du langage C, puisque cela conduirait à allouer un mot pour 1 bit utile pour représenter un lien, plus un mot pour représenter le chaînage.

Nous gérons donc les parois comme des **files de bits contigus** dans des zones mémoire allouées par le système et par incréments de la taille d'un mot par défaut.

Lors de la création d'une paroi, un premier incrément est alloué pour stocker le premier lien; lorsque l'on doit ajouter un nouveau lien à une paroi et que la place allouée est totalement occupée, on adresse une requête au système pour augmenter la taille de la zone mémoire contigue allouée.

La liste des liens formant les contours est gérée de la même façon, à la longueur des liens près qui est alors de deux bits.

4.6. MESURE DES ATTRIBUTS AU FIL DU BALAYAGE.

Les attributs que l'on désire associer à une composante connexe sont les suivants :

- nc: le nombre de ses contours,
- haut, larg: la boite englobante (i.e. la boite englobante de son contour extérieur),
- sp: la somme des périmètres des contours,
- surf: la surface noire (surface intérieure au contour extérieur moins surfaces intérieures aux contours intérieurs).

Le calcul de ces attributs nécessite la mesure pour chaque contour :

- de la surface enfermée,
- de la boite englobante.

4.6.1. Variables incrémentales pour le calcul de la surface.

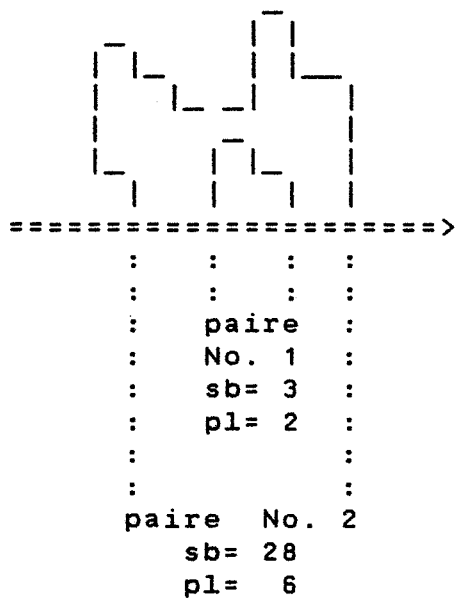
Ce calcul peut être fait de manière incrémentale au cours du balayage en mettant à jour à tout instant,

- la surface intérieure (ou surface balayée sb), et
- la largeur sur la ligne de balayage (ou plage courante pl),

de tous les contours formés par des parois actives et inactives connectées et fermés par la ligne de balayage (cf. fig 2.8).

Ces objets, que l'on pourrait qualifier de composantes connexes actives, s'identifient en fait aux paires de parois actives introduites au paragraphe précédent.

Figure 2.8
variables pour le calcul de surface.



4.6.2. Variables incrémentales pour le calcul de la boite englobante.

Ce calcul nécessite la mise à jour pour chaque paire de parois actives, des grandeurs:

- haut: hauteur actuellement connue,
- cgauche: abscisse gauche actuellement connue,
- cdroite: abscisse droite actuellement connue.

La mise à jour de cgauche et cdroite n'est possible qu'à la condition de disposer à tout instant, pour chaque paroi active, de la coordonnée courante de son point d'intersection avec la ligne de balayage (champ "xp" dans la structure "paroi"), mesurée par rapport à un point de référence: le point de naissance de la paroi, jusqu'au moment où cette paroi atteint un point de fusion

Il faut attendre la fusion de deux parois pour obtenir une référence commune et effectuer la mise à jour des bornes cgauche et cdroite de la nouvelle boite englobante, à partir des boites actuellement connues pour les deux parois. Ces deux anciennes boites peuvent être:

- disjointes, et le point de fusion réalise alors la jonction.
- d'intersection non vide, et il faut alors connaître la position du point de fusion par rapport aux deux boites. Les coordonnées du point de fusion étant connues, on peut alors calculer les nouvelles abscisses absolues cgauche et cdroite.
- l'une et/ou l'autre inconnue(s), dans le cas de la première fusion d'une paroi. On opère alors comme si le point de naissance de la paroi était en dehors de la boite englobante (voir les initialisations).

Le calcul de la hauteur se simplifie par rapport à celui de la largeur en vertu de la propriété suivante:

A chaque nouveau lien 1 (descendant), la hauteur courante d'une paire de parois actives augmente exactement de un. On peut donc associer la hauteur directement à la paire de parois, à condition de ne la mettre à jour qu'une fois pour les deux.

4.6.3. mise à jour des grandeurs incrémentales.

Les mises à jour à opérer lors des opérations lien_0 et lien_1, sont récapitulées ci-dessous :

```
initialisations en un point de naissance:
   xp := 0 sur la paroi gauche
   xp := 1 sur la paroi droite
   haut, pl := 1
   sb, cdroite := 0
   cgauche := INFINI

lien_0 sur une paroi gauche:
   xp := xp + 1  abscisse courante de la paroi
   pl := pl - 1  distance entre les 2 parois actives
```

```
lien_0 sur une paroi droite:
   xp := xp + 1
   pl := pl + 1

lien_1 sur une paroi gauche:
   haut := haut + 1

lien_1 sur une paroi droite:
   surf := surf + pl
```

4.6.4. Mise à jour aux points de mort.

4.6.4.1. Calcul de la surface.

La mise à jour de sb et pl dépend du cas de fusion qui se présente:

```
fusion g-d:
   surf := sb

fusion d-g:
   sb := sb(1) + sb(2)
   pl := pl(1) + pl(2)

fusion g-g:
   sb := sb(1) - sb(2)
   pl := pl(1) - pl(2)

fusion d-d:
   sb := sb(2) - sb(1)
   pl := pl(2) - pl(1)
```

4.6.4.2. Calcul de la boite englobante.

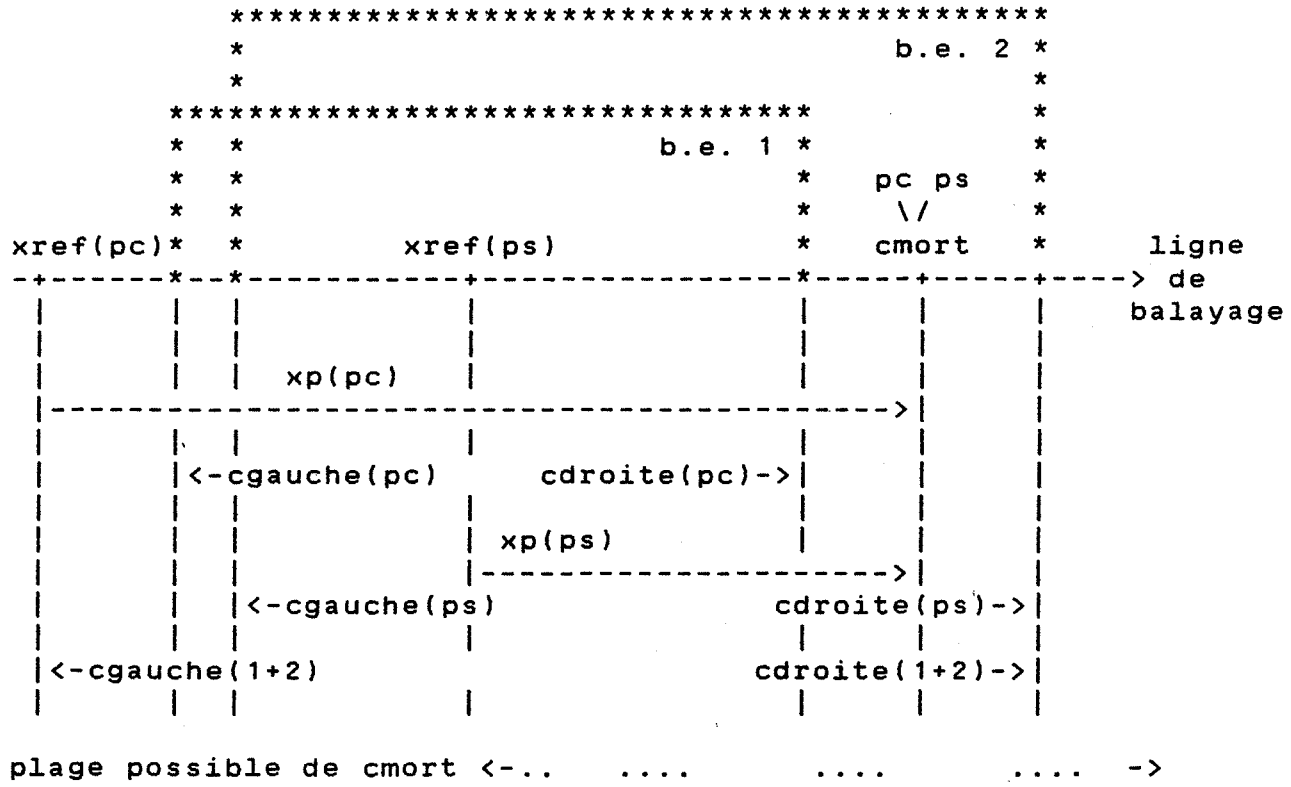
Largeur:

Pour chaque paire de parois "cgauche" et "cdroite" désignent les abscisses absolues de la boite englobante connue jusque là. A chaque point de mort (seul point dont on possède les coordonnées absolues), il faut mettre à jour ces coordonnées de la façon suivante:

```
[cgauche , cdroite] =
   [cgauche(pc) , cdroite(pc)]
 U [cgauche(ps) , cdroite(ps)]
 U [min (cmort - xp(pc) , cmort - xp(ps)) , cmort]
```

où cmort désigne la colonne du point de mort.

Figure 2.9
mise à jour de la boite englobante.



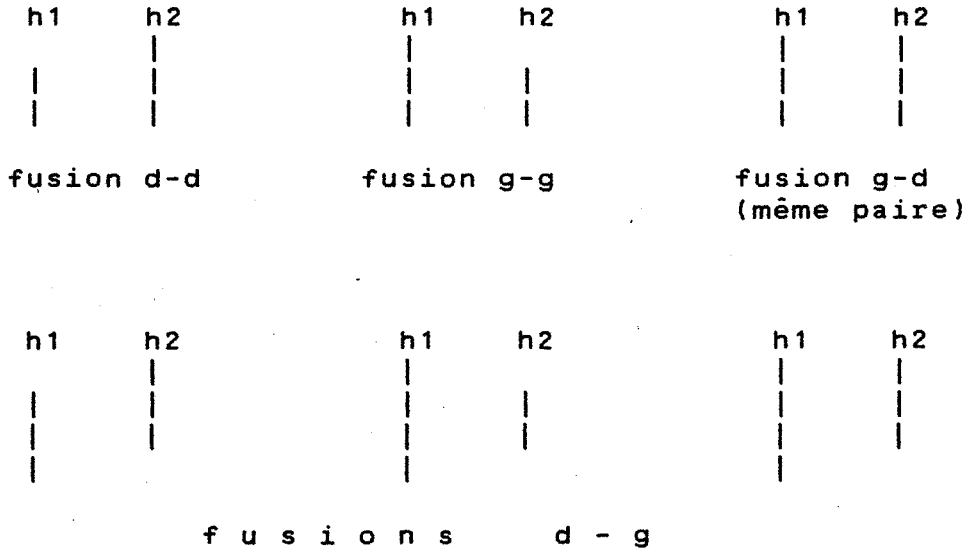
Si le point de mort ferme un contour externe, la largeur de boite englobante est "cdroite - cgauche" après la mise à jour des coordonnées en ce point.

Hauteur:

Pour chaque paire de parois on a la hauteur connue de la paire à partir du début de vie de cette paire, et à travers toutes les fusions qu'elle a pu subir.

En un point de mort il faut comparer les hauteurs associées à chacune des parois, qui représentent les hauteurs actuellement connues des composantes connexes auxquelles appartiennent (séparément avant le point de mort) chacune des parois pc et ps. Soient h1 et h2 ces hauteurs. Leurs positions relatives sont données par la figure suivante.

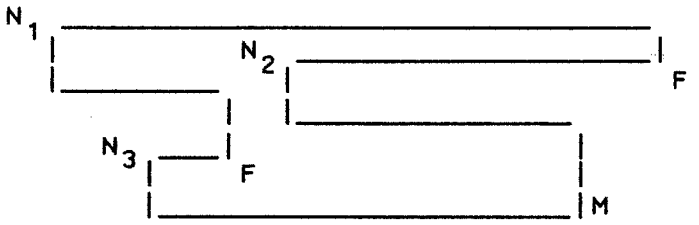
Figure 2.10
calcul de la hauteur



Dans tous les cas de fusion droite-gauche la hauteur de la paire associée à la paroi gauche a été mise à jour au balayage de la ligne précédente, alors que la hauteur de la paire associée à la paroi droite l'a été à la ligne actuelle. Ceci provoque l'asymétrie de ce cas par rapport aux autres. Mises à jour à effectuer:

fusion droite-gauche: haut := max(h2+1 , h1)
tous les autres cas : haut := max(h1 , h2)

Figure 2.11
Exemple résumant les problèmes de largeur et hauteur.



N: point de naissance
F: point de fusion
M: point de mort

Dans les points de fusion, la hauteur des parois gauches est déjà 1. Déplacer les deux tranches horizontales l'une par rapport à l'autre pour avoir toutes les possiblités de la mise à jour de la largeur.

5. *AFFICHAGE SUR BITMAP.*

5.1. GENERALITES.

A partir des contours codés des différentes composantes connexes de l'image, on peut procéder de plusieurs façons distinctes pour reconstruire une image complète.

1. Reconstruire une image constituée d'un ensemble de contours puis utiliser des algorithmes classiques de remplissage de contours.

2. Utiliser un algorithme qui exploite la structure des codes de contours de l'image et fournit en un seul balayage ligne après ligne les plages noires et blanches.
   Cet algorithme peut se comprendre comme une sorte d'algorithme inverse, mais simplifié, de celui mis en oeuvre dans le module d'extraction des composantes connexes.

3. Reconstruire une image constituée d'un ensemble de contours par un algorithme qui exploite la structure des codes des contours et quelques fonctions très classiques implantées sur les terminaux graphiques à balayage de trame (dits "raster").

La première solution est très intéressante lorsque le terminal de consultation possède une certaine intelligence (éventuellement des fonctions cablées ou microprogrammées de remplissage de contours, de traçés de vecteurs...). Comme il n'existe pas de spécifications uniformes pour ces terminaux nous n'avons pas réalisé de module correspondant à ce cas, mais l'intégration de fonctions évoluées devrait simplifier la tâche dans les algorithmes que nous présentons.

Nous allons décrire dans ce qui suit, d'une part le fonctionnement d'un module d'affichage utilisant le deuxième principe, appelé **affichage par balayage** puis un algorithme d'affichage de contours, dit **affichage aléatoire** utilisant une fonction classique des terminaux graphiques.

Puisqu'une image est constituée de composantes connexes de natures distinctes, il est naturel de prévoir des algorithmes d'affichage adaptés à des classes d'objets différents. Ces algorithmes devront profiter des propriétés des codages des objets.

L'affichage par balayage utilise comme seule fonction d'affichage du terminal, la fonction très simple:
plage(ièmeligne,colonnedébut,colonnefin) qui affiche sur l'écran à la ligne numéro i, une plage de pixels, de la colonne colonnedébut à la colonne colonnefin

(incluses).

L'affichage aléatoire utilise en plus de cette fonction le mode d'accès OU-Exclusif qui effectue un "XOR" entre les bits de la mémoire d'image et ceux envoyés par cette fonction.

5.2. RECONSTRUCTION PAR BALAYAGE.

5.2.1. Spécifications.

DECODAGE D'UNE LISTE DE COMPOSANTES CONNEXES.

ENTREE : une structure de données de composantes connexes ordonnées selon l'ordre lexicographique inverse de leur dernier point de mort;

SORTIE : l'image binaire reconstruite.

Le décodage est réalisé au fur et à mesure d'un balayage unique, ligne après ligne, de DROITE à GAUCHE et de BAS en HAUT, de l'image à reconstruire. On procède par activations successives de composantes connexes, puis de contours, et enfin de parois.

L'algorithme utilise les structures de données suivantes :

- liste des composantes connexes, c'est à dire le code lui-même de l'image;
- liste des contours (intérieurs, non encore actifs, des CCX actives);
- liste des couples de parois inactives (des contours actifs)
- anneau des parois actives;
- tableau d'indicateurs relatifs à chaque interpixel de la ligne de balayage, indiquant le nombre de parois actives en cet interpixel (0, 1 ou 2 parois actives). Selon la convention de connexité, le cas de deux parois actives en un interpixel est possible ou non.

Les listes sont ordonnées selon l'ordre lexicographique inverse des points de départ de leurs éléments.

A tout instant, le couple (l, c) désigne :
- l'interpixel courant,
- le pixel que l'on va colorier.

Remarque:

On pourrait être tenté de réaliser le décodage en 2 étapes inverses des 2 étapes du codage (précode, ccx). Un décodage, inverse de "précode", est effectivement facile à mettre au point. Par contre, reconstruire le précode à partir du code de contour (inverse de ccx) n'est pas facile. Si l'on y arrive c'est à l'aide d'un algorithme qui permet, sans travail supplémentaire autre que l'écriture en mémoire d'image, de reconstruire l'image; dès lors pourquoi restreindre ce mécanisme et introduire une étape supplémentaire ?

5.2.2. ALGORITHME DE RECONSTRUCTION.

```
/*                AFFICHAGE PAR BALAYAGE                              */
/* lx et cx représentent la taille de l'image a reconstruire */

/* INITIALISATIONS */
lccx, cccx := lecture des coord du pt de depart de la 1ere CCX
init_couples();
init_contours();
pp:=ps:=pc:=Nil;
coul := BLANC;

/* Lecture des index permettant de récupérer les ccx
    dans l'ordre lexicographique inverse   */
lire (fichier index);

/* ON BALAYE L'IMAGE ENTIERE POUR LA RECONSTRUIRE */
l:=lccx;
tant que ( l >=0 )

    pour c := cx jusqu'à 1 faire
        si (pactives [c] > 0) alors
            si (pactives [c] = 2) alors
                ABSORBER_LIEN;
                ABSORBER_LIEN
            sinon
                ABSORBER_LIEN
            finsi
        sinon
            si (ATTEINT(lcple,ccple)) alors
                ACTIVER_COUPLE
            sinon
                si (ATTEINT(lcont,ccont)) alors
                    ACTIVER_CONTOUR
                sinon
                    si (ATTEINT(lccx,cccx)) alors
                        ACTIVER_CCX;
                    finsi
                finsi
            finsi
        finsi
    finpour

    /* s'il y a des lignes blanches on saute à la ccx suivante.
      abs(lcple)=INFINI lorsqu'il n'y a plus de couples en
      attente.  pc=Nil lorsqu'il n'y a plus de parois actives. */

    si ( (pc = Nil ) et ( abs(lcple) = INFINI ) ) alors
        l:=lccx;
    finsi

fintant


ATTEINT (ligne,colonne) est vrai lorsque ligne = l
                                    et colonne = c ;
```

Justification de l'algorithme.

Cet algorithme utilise quatre procédures de base:
ACTIVER_CCX
ACTIVER_CONTOUR
ACTIVER_COUPLE
ABSORBER_LIEN

L'activation d'une ccx entraine l'activation de son contour extérieur. L'activation d'un contour entraine l'activation du couple correspondant au dernier point de mort du contour. L'activation d'un couple entraine l'absorption de deux liens: un lien 1 sur la paroi droite et un lien 0 sur la paroi gauche. L'absorption d'un lien vertical provoque un changement de la couleur d'affichage.

5.2.3. Procédures d'activation.

A chaque activation d'une composante connexe les contours qui la composent forment une sous-liste de contours inactifs qui est fusionnée avec la liste globale des contours inactifs. La composante connexe perd alors toute consistance au profit d'un ensemble de contours dont on ignore l'origine.

De même, l'activation d'un contour entraîne une décomposition de ce contour en sous-listes de couples de parois, chaque couple étant composé d'une paroi droite et une gauche. La sous-liste est fusionnée avec la liste globale des parois inactives et le contour générateur est éliminé.

Algorithmes correspondants:

```
ACTIVER_CCX

lire ccx;              (* on lit tous les codes des contours *)
ACTIVER_CONTOUR;             (* contour extérieur *)
construire une sous-liste ordonnée des contours intérieurs;
                             (* ordre l,c inverse *)
fusionner la sous-liste avec la liste globale
                             des contours inactifs;

lcont:=pcont->l;         (* mise à jour du prochain contour*)
ccont:=pcont->c;             (* intérieur à activer*)
                          (* pcont: pointeur sur contours *)
lire lccx,cccx;              (* de la ccx suivante *)
```

```
ACTIVER_CONTOUR

décomposer le contour en couples de parois;
                                    (* repérées droite gauche *)
ACTIVER_COUPLE;                       (* premier couple *)
construire une sous-liste ordonnée des parois;
                (* ordre l,c inverse et droite puis gauche *)
fusionner la sous-liste avec la liste des parois inactives;

lcple:=pcple->l;              (* mise à jour de la prochaine *)
ccple:=pcple->c;                  (* paroi à activer *)
                               (* pcple: pointeur sur couple *)
pcont := pcont ->sv;           (* passage au contour inactif *)
lcont := pcont ->l;                     (* suivant *)
ccont := pcont ->c;
```

Remarque:

La notion de couple de parois n'est nécessaire que pour l'ordre de construction de la sous-liste. Elle n'a aucune raison d'être dans la suite, et nous utilisons le terme "couple de parois" uniquement pour signaler que les parois seront toujours activées deux par deux.

```
ACTIVER_COUPLE

transférer les deux premières parois de la liste des
          parois inactives dans la liste des parois
          actives avant la paroi courante;
pactives[c] := 2;               (* il y a 2 liens à consommer *)
ABSORBER_LIEN;
ABSORBER_LIEN;

pcple := pcple ->sv->sv;    (* passage aux parois inactives *)
lcple := pcple ->l;                   (* suivantes *)
ccple := pcple ->c;
```

5.2.4. Construction des parois à partir du code de contour.

Les contours sont décomposés en suivant des parois "montantes" (directions 2 et 3) et "descendantes" (directions 0 et 1). Cette distinction permet de suivre les parois, mais n'est pas suffisante pour déterminer l'ordre droite-gauche.

Une paroi **droite** est soit une paroi montante qui commence par un lien vertical (direction 3), soit une paroi descendante qui se termine par un lien vertical (direction 1). Une paroi **gauche** est soit une paroi montante qui commence par un lien horizontal (direction 2), soit une paroi descendante qui se termine par un lien horizontal (direc-

tion 0).

5.2.5. Vie et mort d'une paroi dans l'anneau.

La consommation des liens d'une paroi obéit aux règles suivantes:

1. Un lien 1 transmet l'activité vers le haut sauf si la paroi est terminée. On aura donc un lien à consommer à la ligne suivante, même colonne, sauf si la paroi est entièrement consommée. Pour la ligne en cours il faut changer de couleur et passer à la paroi suivante.

2. Un lien 0 transmet l'activité vers la gauche sauf si la paroi est terminée. On continuera donc la consommation de la même paroi en transmettant l'activité à l'inter-pixel suivant de la même ligne sans changer de couleur.

Algorithme correspondant:

```
ABSORBER_LIEN

finparoi := FAUX;
si (dernier_lien)  alors  finparoi := VRAI;

(* inversion de la couleur à chaque lien vertical *)
si (lien = 1) alors
    si ( couleur = NOIR ) alors
        afficher (l,c+1,cprec)
    sinon
        couleur := non (couleur);
        cprec:= c
    finsi
finsi;


(* activation/desactivation sur les colonnes*)
si (lien = 0) alors
    pactives[c] := pactives[c] - 1;
    pactives[c-1] := pactives[c-1] +1
finsi;
```

```
(* desactivation en fin de paroi *)
si (finparoi)  alors
    si (lien=1) alors
        pactives[c]:=pactives[c] - 1
    sinon
        pactives[c-1]:=pactives[c-1] -1
    finsi
    détruire la paroi courante;
sinon
    si (lien = 1) alors
        avancer d'une paroi
    finsi
finsi
```

```
AVANCER d'une paroi:
pp := pc;
pc := ps;
ps := pc->sv;
```

```
DETRUIRE la paroi courante:
pc := ps;
pp->sv := pc;
ps := pc->sv;
ps:=pc->sv;
```

5.3. AFFICHAGE "ALEATOIRE", OU ALGORITHME D'AFFICHAGE DE CONTOURS.

5.3.1. Spécifications.

ENTREE: Un ensemble de contours non nécessairement ordonné.

SORTIE: l'image binaire reconstruite.

Le décodage est réalisé contour par contour. L'algorithme utilise uniquement une structure de pile.

Nous supposons que le terminal d'affichage possède une mémoire au moins égale à la taille de l'écran et que la fonction XOR est implémentée. Cette capacité est nécessaire pour pouvoir superposer des contours.

Ces spécifications ne sont pas hors du commun dans les terminaux à balayage par trame (rase terre).

5.3.2. ALGORITHME DE RECONSTRUCTION.

```
/*              AFFICHAGE  ALEATOIRE                              */
/* initialiser le mode de fonctionnement du terminal en XOR */
initbitmap;
tant que ( non (fin de fichier) )faire
     lire (contour);
     traiter_contour
fintant



procédure traiter_contour;

lire lcont,ccont,longueurcont;/* coordonnées et longueur contour*/

colonact:=ccont;
lignact:=lcont;
pour j:=1 jusqu'à longueurcont faire
     extraire(lien);
     selon (lien)
          0: colonact := colonact + 1;
          2: colonact := colonact - 1;
          3:
               empiler (colonact);
               lignact := lignact - 1;
          1:
               lignact := lignact + 1;
                dernierecol := haut_de_pile;
               si (colonact > dernierecol) alors
                    plage (lignact,dernierecol+1,colonact)
                 sinon
                    plage (lignact,colonact+1,dernierecol)
               finsi
               depiler
     finselon
finpour
```

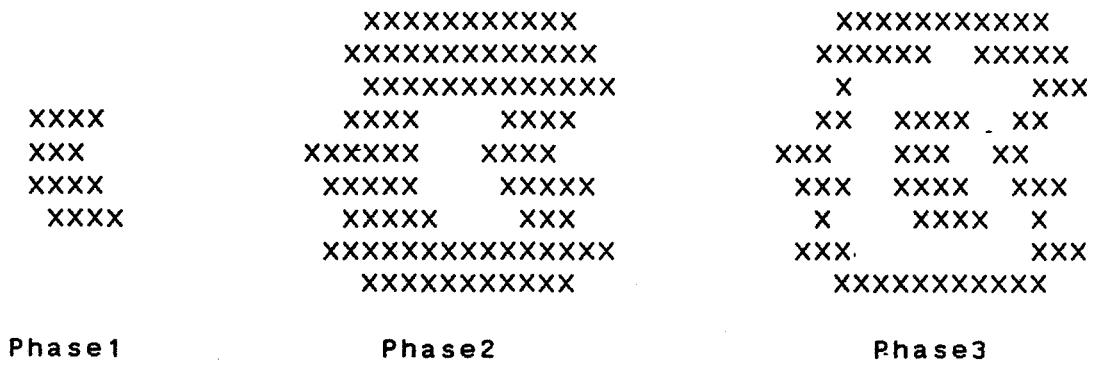
5.3.3. Principes utilisés dans cet algorithme.

1. Lorsqu'une composante connexe ne contient aucune autre composante connexe, le contour extérieur est suivi et rempli de noir (en fait de la couleur opposée à celle du fond). Les contours intérieurs sont ensuite suivis selon le même procédé et sont affichés avec la couleur opposée à celle qui y est déjà, donc opposée à celle du contour extérieur. Cette opposition est engendrée par le mode XOR. Si on commence par les contours intérieurs, ils sont d'abord "noircis", puis "blanchis" lors de l'affichage du contour extérieur.

2. Lorsqu'une composante connexe C1 contient une autre composante connexe C2, cette dernière ne peut qu'être incluse dans un contour intérieur de la première.
   Si C2 a déjà été affichée elle sera inversée deux fois (avec le tracé extérieur de C1 puis avec le tracé du

contour intérieur de C1 qui englobe C2). Si C2 est affichée après C1 elle vient s'opposer dans le contour intérieur qui l'englobe. Si C2 est affichée après le contour extérieur de C1 elle se comporte dans un premier temps comme un contour intérieur à C1 puis est inversée lors de l'affichage du contour intérieur l'englobant. Ce raisonnement reste valable si seul le contour intérieur de C1 a été affiché.

Figure 2.12
Affichage d'une composante incluse dans une autre.



Phase1: affichage de la composante connexe "intérieure"; le fond est blanc; la composante connexe est noire.

Phase2: le contour extérieur de la composante connexe "extérieure" est affiché; le fond est blanc; la composante connexe "intérieure" est inversée et devient blanche.

Phase3: le contour intérieur de la composante connexe "extérieure" est affiché; il provoque l'inversion des pixels qui le composent (noirs actuellement) dans la composante connexe "extérieure" à laquelle il appartient, et des pixels blancs de la composante connexe "intérieure" qu'il enveloppe.

Remarque:

On constate qu'il n'y a pas d'ordre privilégié d'affichage ni entre les composantes connexes, ni entre les contours d'une même composante connexe, ni entre les divers contours de plusieurs composantes connexes. On peut aussi bien afficher d'abord quelques contours intérieurs puis le contour extérieur et ainsi de suite.

3. Pour afficher un contour en le suivant, cet algorithme procède par accouplement de liens 3 et 1 sans chercher les couples appariés. Une paire 3-1 est appariée lorsqu'elle ne contient aucune autre paire (elle ne

peut contenir un nombre impair de liens verticaux). Une paire non appariée est incluse dans une autre paire non appariée. Les liens 0 et 2 permettent le calcul des déplacements horizontaux uniquement. Les liens 3 sont empilés; les liens 1 provoquent l'affichage avec le 3 du haut de la pile, et le dépilement. Il reste à prouver que cette démarche est correcte.

5.3.4. Preuve de l'algorithme.

D'afficher en mode XOR permet d'écrire un algorithme qui procède **par affichage de contours en oubliant s'il s'agit de contours intérieurs ou extérieurs.**

Une composante connexe peut ainsi se voir comme une **union** de composantes connexes constituées d'un seul contour extérieur à afficher **dans un ordre quelconque.**

Sans perte de généralité nous allons donc étudier cet algorithme sur une composante connexe constituée d'un seul contour extérieur, et en faire la preuve par induction sur la longueur n du contour (en nombre de liens) de la composante connexe à afficher.

L'algorithme en pidgin devient alors:

En parcourant le contour tant qu'il existe un lien:

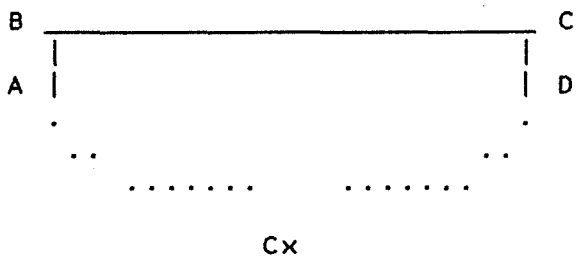
- Ne rien faire pour les liens horizontaux - Empiler les liens montants - A chaque lien descendant afficher en XOR la plage comprise entre le haut de la pile (le dernier lien montant) et ce lien descendant, et dépiler.

Sur la plus petite composante connexe de surface 1 et dont le contour a pour longueur 4, le résultat est immédiat car il faut bien afficher exactement une plage.

Supposons l'algorithme valide pour des composantes connexes dont le contour a une taille $< n$.

Soit Cx une nouvelle composante dont le contour a une longueur de n. Au premier dépilement (il en existe nécessairement un, car un contour est une courbe fermée du plan) on se trouve dans la situation de la figure suivante.

Figure 2.13



(où A,B,C,D représentent des interpixels, et AB,CD des liens verticaux).

Considérons Cx' le nouveau contour tel que: Cx = Cx' ⊕ AD somme booléenne de contours (i.e. si Cx' ∩ AD ≠ ∅ alors Cx est égal à leur union disjointe).

Par hypothèse d'induction, comme longueur(Cx') < n, l'algorithme affiche bien Cx'.

Il est clair que pour passer de Cx' à Cx, il suffit de faire un affichage en XOR de la plage [A,B,C,D].

□

Analyse en complexité:

a)

Si l'on compte en O(1) l'affichage en XOR d'une plage, l'algorithme est alors trivialement linéaire en fonction de la longueur du contour.

b)

Manifestement cette évaluation n'est pas licite (sur le bitmap NUMELEC, ces fonctions sont programmées en assembleur et prennent un temps certain).

Cependant, soit Nbaffich le nombre total d'affichages en XOR requis par l'affichage d'un contour. La majoration suivante est triviale:

Nbaffich < Hauteur . Nmax

où Nmax est le nombre maximum de paires d'intersections d'une ligne de pixels de l'écran avec le contour.

Bien sûr nous avons:

Nbaffich < Hauteur . Largeur

(mais alors l'évaluation de l'algorithme devient quadratique)!

En revanche remarquons que pour un symbole (composante connexe de la taille d'un caractère) Nmax ne doit pas être trop grand. En effet pour tous les caractères de l'alphabet ce nombre est inférieur à quatre (par exemple pour les lettres M, W et @) avec le contour.

Ceci justifie pleinement l'utilisation de cet algorithme pour afficher des symboles représentés par leurs contours.

c)

Dans le cadre d'une analyse plus fine, considérons le nombre d'opérations élémentaires requises par l'affichage d'une plage comme une fonction linéaire du nombre de pixels (i.e. de la longueur de la plage).

La fonction Faffich de complexité associée devient:

$$Faffich = c \cdot \Big(\sum_{AB,CD \text{ paire de liens verticaux appariés par l'algorithme.}} | [A,B,C,D] | \Big)$$

$$\leqslant c.\ Hauteur.\ Largeur.\ Nmax$$

$$\leqslant c.\ Surf\ .Nmax$$

où Surf est la surface du rectangle englobant de la composante connexe considérée.

D'après cette formule, il est clair que cet algorithme est pénalisé car il peut inverser plusieurs fois la couleur d'un même pixel de l'écran. En effet l'algorithme présenté n'affiche pas exclusivement des plages intérieures à une composante connexes.

Ainsi pour les figures compliquées telles une coquille d'escargot (sorte de plus mauvais cas, car les pixels de la zone centrale de la coquille sont affichés exactement Nmax fois) il est préférable d'écrire un algorithme qui n'affiche qu'à coup sûr en mode d'affichage normal.

Nous ne l'avons pas encore réalisé mais ceci est naturellement possible. Encore faudrait-il s'assurer que ce deuxième algorithme ne soit pas pénalisé dans les traite-

ments hors affichage, puisque nous n'avons analysé que la complexité de l'affichage pur.

Là encore cet algorithme semble le bon compromis pour l'affichage des symboles dont la surface du rectangle englobant est par définition faible.

Variantes.

Il existe une variante naturelle de cet algorithme qui permet d'exhiber la structure de parenthésage sous-jacente à un contour, et que l'on peut résumer comme suit:

En parcourant le contour à chaque lien vertical (montant ou descendant) affficher en XOR la partie située à gauche du lien de la ligne de l'écran contenant ce lien vertical.

Il suffit de choisir son bord (gauche ou droite) et de s'y tenir.

Malheureusement cet algorithme dont l'expression est encore plus simple -car dans ce cas il n'est même pas nécessaire d'empiler les liens montants- est notoirement plus lent que le précédent.

Remarquons en outre que cet algorithme permet de traiter les parois indépendamment les unes des autres. Nous n'avons pas exploité cette possibilité, mais cela peut être utile pour certaines variantes du codage à développer ultérieurement.

Commentaires sur cet algorithme:

Cet algorithme est à ranger dans la classe des algorithmes dits de parité (cf. PAVLIDIS [PAVL]). Son expression est ici particulièrement simple car elle utilise le codage de contour développé au début de ce chapitre.

Généralisation.

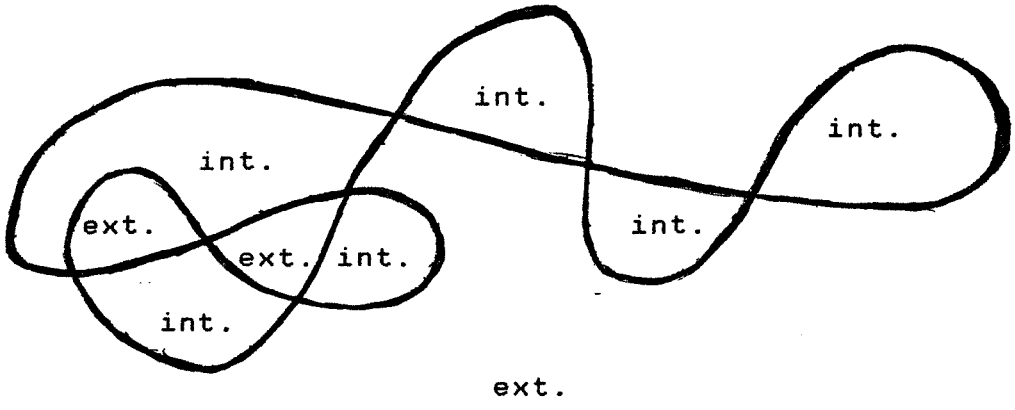
Quelques définitions.

Considérons une courbe fermée continue du plan C; nous allons rappeler de quelle manière on peut lui associer un intérieur et un extérieur.

Soit G_C le graphe planaire dont les sommets sont les points du plan où la courbe se coupe elle-même. Deux sommets x,y de G_C sont reliés par une arête de G_C, s'il existe un morceau de la courbe C ne contenant pas d'autre sommet que x et y et les joignant.

Remarque:

Ce graphe peut contenir des boucles.



Par construction ce graphe G_C est eulérien (i.e. il existe une chaîne qui emprunte une fois et une seule chaque arête). En outre la propriété suivante est bien connue.

Proposition 2.1:

Les faces d'un graphe planaire eulérien sont bicolorables.

Preuve:

Il suffit de vérifier que le graphe dual de G_C est sans cycle impair (i.e. biparti). □

Ainsi par convention nous appelerons extérieur de C l'ensemble des points situés à l'intérieur des faces de G_C de la même couleur que celle de la face à l'infini. De même l'intérieur se définit comme l'ensemble des points situés à l'intérieur des autres faces de G_C.

Théorème 2.1:

(variante du théorème d'Euler [EULE])

Soit C une courbe fermée du plan donnée par son code de contour, l'algorithme précédent d'affichage de contours, appliqué au contour de C inverse la couleur de la partie intérieure de C.

Preuve:

La preuve est identique à celle déjà proposée pour l'affichage d'une composante connexe faite précédemment. □

Corollaire 2.1:

On peut traiter une image contour par contour.

Corollaire 2.2:

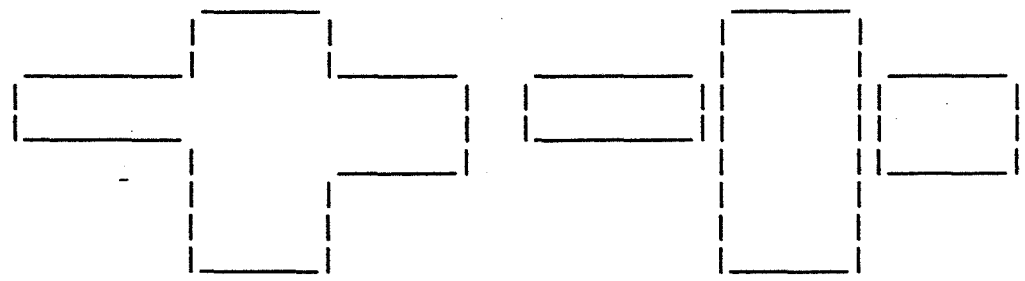
Toute partition d'un contour C en contours disjoints* C_i ($C = \oplus C_i$) est licite pour l'affichage.

Dans l'implémentation décrite au chapitre 4, nous avons abondamment utilisé le corollaire 2.1. En particulier, dans notre algorithme d'élimination des "barbules" (sorte d'aspérités) qui procède contour par contour, nous ne nous soucions pas du fait que les transformations apportées font que deux contours se chevauchent.

Par contre, nous n'avons pas utilisé le deuxième corollaire.

Figure 2.14

exemple de partition d'un contour:



* les contours peuvent avoir des arêtes communes mais leurs intérieurs n'ont pas de pixels communs (cf. figure).

CHAPITRE 3

LES CODES DE CONTOURS MANIPULATION ET CODES OPTIMAUX

Dans ce chapitre nous définissons des classes de composantes connexes ou contours selon les attributs des objets. Nous cherchons ensuite à caractériser l'ensemble des suites de liens représentant des contours, puis nous présentons des algorithmes de compression adaptés à chaque classe.

1. REPRESENTATION D'UNE IMAGE PAR CODES DE CONTOURS.

Après la phase de détection par balayage décrite au chapitre précédent, se pose maintenant le problème du choix d'un codage. Les solutions proposées ci-après ont été choisies afin de minimiser la longueur du codage transmis (recherche du taux de compression maximum).

Toutefois, nous ne considérons que des codages exacts, à savoir ceux où chaque contour reste nécessairement fermé. Ainsi, en envisageant des manipulations (ou déformations) sur les contours, nous proposerons systématiquement la conservation de cette propriété.

Pour apprécier les différences entre les codes proposés, nous présenterons les résultats de notre application (cf. chapitre suivant).

Le problème du taux de compression maximal est difficile et empiète de manière non négligeable sur celui de la reconnaissance des formes. Les codes optimaux sont trivialement ceux qui se servent de la reconnaissance de l'objet à coder. Nous ne traiterons pas cet aspect dans ce chapitre; signalons toutefois qu'une façon d'aborder ce problème par la constitution d'une bibliothèque de contours est abordée dans le chapitre suivant.

Le sous-problème que nous considérons ici peut se définir comme suit:

donnée: une liste de contours d'interpixels de type (ligne, colonne) (suite de liens), où ligne et colonne sont les coordonnées du point de mort du contour, celui-ci étant décrit dans le sens négatif trigonométrique à partir du point de mort par la suite de liens.

résultat: un codage exact des contours de l'image.

Ligne et colonne dépendent de la taille de l'image, et un codage optimal devra en tenir compte. Par contre la suite de liens ne changera pas de nature, quelle que soit la taille de l'image ou le facteur de résolution. Nous l'étudions dans tous ses détails ici.

2. DETECTION D'OBJETS PARTICULIERS.

2.1. TYPES DE COMPOSANTES CONNEXES.

Les attributs des composantes connexes permettent de classer les objets en définissant des paramètres dépendants de l'application. Lorsque l'entité "composante connexe" est importante on peut utilser les définitions suivantes:

2.1.1. Symbole.

Une composante connexe est considérée comme un symbole si son rectangle englobant est contenu dans un rectangle maximal fixé (défini par deux paramètres Largeur et Hauteur).

2.1.2. Objet mince.

Une composante connexe est considérée comme mince si sa surface noire est du même ordre que son périmètre (i.e. Surf / Per < 2+Epsilon où Epsilon est un paramètre du module de sélection).

2.1.3. Résidu.

Les composantes connexes ne satisfaisant pas à l'un des deux critères précédents sont des moins que rien autrement dit des résidus.

2.1.4. Rejets possibles.

Lorsqu'une composante connexe n'atteint pas une taille minimale fixée elle peut être rejetée. Cette taille est de l'ordre de 6 pixels (pour un facteur de résolution de 8 points/mm), mais il faudra, en fonction des résultats recherchés (précision de la reproduction) affiner cette dimension. Un point dans un texte par exemple est souvent de dimension 3x3.

Question:

Si l'on considère comme défaut de l'image une petite tache noire, ne doit-on pas réagir de même sur une petite tache blanche (intérieure à un contour)?

Il suffit dans notre cas de prendre comme critère d'élimination le périmètre du contour, au lieu de procéder par fenêtrages successifs. Par exemple, choisir comme périmètre éliminatoire 10, revient à rejeter la plupart des contours inclus dans une fenêtre 3x2 ou 4x1.

On peut ainsi envisager ces rejets sur tous les contours, i.e. non seulement pour les composantes connexes, mais aussi pour **les contours intérieurs éventuels** n'atteignant pas cette taille minimale. Ces rejets imposent alors de modifier les caractéristiques de la composante connexe les contenant, à savoir:

- nombre de contours := nombre de contours - 1 pour chaque contour intérieur éliminé

- surface noire := surface noire + surface du contour éliminé.

Il est tout à fait licite de rejeter ces contours lorsqu'on choisit la 8-connexité, car il n'existe pas d'autres objets dans le voisinage et on peut considérer ces contours comme non significatifs. Dans les autres cas on risque d'éliminer successivement des petits contours voisins, dont l'ensemble peut être significatif.

2.2. TYPES DE CONTOURS.

Le choix de travailler sur les contours plutôt que sur les composantes connexes peut être intéressant lorsque la distinction intérieur/extérieur n'est pas recherchée. Par ailleurs, ces deux modes de travail ne sont pas exclusifs. En effet, lorsqu'une composante connexe a été classée comme objet mince, elle peut contenir des contours intérieurs de la taille d'un symbole par exemple. Dès lors, pourquoi ne traiterait-on pas ce contour comme un symbole à part entière? De plus, avec les deux techniques d'affichage que nous avons présentées nous pouvons aussi bien dissocier ces contours, que conserver l'inclusion.

2.2.1. Minis-contours.

L'énumération de tous les contours de faible longueur nous paraît possible et intéressante. Elle permettrait de remplacer la description des liens du contour par une référence, et de provoquer une génération directe à l'affichage. Nous en donnons quelques éléments à la fin de ce chapitre (paragraphe "codes optimaux de suites de liens").

2.2.2. Petits contours.

Ce sont les contours qui vont entrer dans un module de constitution de bibliothèque. Typiquement, cette classe contient les contours des "symboles" définis ci-dessus, et une partie des contours intérieurs aux "grands" contours.

Nous effectuons la séparation entre "petits" et "grands" par la longueur du contour (de l'ordre de 2 fois le périmètre de la fenêtre d'un symbole).

2.2.3. Grands contours.

Nous mettons dans cette classe les contours pour lesquels:

- il est peu probable de trouver des similitudes, ces contours étant peu nombreux et plutôt dissemblables,

- on envisage un traitement sur des tronçons du contour (voir le paragraphe factorisation dans ce chapitre).

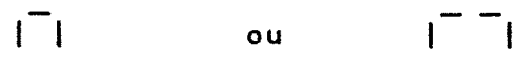
En effet, plus les contours sont grands moins ils sont nombreux et il y a peu de chances alors pour que la comparaison de leur codes de contours globaux donne des résultats positifs. Il faut remarquer ici qu'un trait en travers d'une page transforme toutes les composantes connexes qu'il coupe en une et une seule composante. On obtient alors un contour extérieur plutôt "difforme" et des contours intérieurs qu'une bibliothèque de contours pourra éventuellement sauver.

2.3. BARBULES.

2.3.1. Définition:

On appelle **barbule** d'un contour un morceau de contour de longueur 3 ou 4 entourant une surface d'un ou deux pixels entre deux liens opposés.

Figure 3.1
exemple de barbules.



Ces aspérités ou barbules peuvent provenir soit du processus de binarisation de l'image, soit tout simplement de la mauvaise qualité de l'image initiale.

Leur élimination permet d'améliorer les performances de la vectorisation des contours, sans nuire à la qualité de l'image (bien souvent le contraire).

L'algorithme suivant élimine en un seul balayage des contours l'essentiel des barbules.

Remarquons que le traitement s'applique de manière identique sur les contours intérieurs ou extérieurs.

Lors de l'implémentation de ce programme, nous sommes tombés sur le problème suivant:

Il existe des objets de surface aussi grande que l'on veut et d'"épaisseur nulle", comme l'indique la figure suivante.

Figure 3.2
objets d'"épaisseur nulle".

chaque rectangle contenant un ou deux pixels.

Que faire d'un tel objet ?

Nous avons considéré que de tels objets étaient significatifs (bien que leur existence soit liée à une mauvaise binarisation ou à une mauvaise qualité de l'image initiale) et donc nous avons modifié l'algorithme d'élimination des barbules pour leur donner une épaisseur de 1. Il suffit pour ce faire de conserver les 3 liens du bout de cette chaîne, lorsque l'élimination du dernier rectangle ou carré provoquerait une "vraie" épaisseur nulle (i.e deux liens consécutifs opposés).

2.3.2. Algorithme d'élimination des barbules.

```
/*        Barbules          */

Pour chaque contour
lire(lien_a,lien_b);

tant qu'il existe un lien faire

  si lien_a=lien_b  alors
       écrire(lien_a);
       lien_a:=lien_b;
       lire(lien_b)
  sinon
       lire(lien_c);
       si lien_a=opposé(lien_c) alors          /* cas |_|   */
           lire(lien_d);
           si lien_b=opposé(lien_d) alors
               carreferme
           sinon
               écrire(lien_b);
               lien_a:=lien_d;
               lire(lien_b)
           finsi
       sinon
           si lien_c=lien_a   alors
               écrire(lien_a);
               lien_a:=lien_b;
               lien_b:=lien_c
           sinon
               lire(lien_d);
               si lien_a=opposé(lien_d) alors   /* cas |__| */
                   lire(lien_e);
                   si lien_c=opposé(lien_e) alors
                       lire(lien_f);
                       si lien_b=opposé(lien_f) alors
                           rectangleferme
                       sinon
                           écrire(lien_b);
                           lien_a:=lien_f;
                           lire(lien_b)
                       finsi
                   sinon                  /*                    _    */
                       chercher_affreux /* recherche de    _|  |   */
                   finsi                  /*                |_ _|   */
               sinon
                   écrire(lien_a,lien_b);
                   lien_a:=lien_c;
                   lien_b:=lien_d
               finsi
       finsi
  finsi
fintant
finpour
```

2.4. ESCALIERS.

Nous avons trouvé intéressant de distinguer des morceaux de contour ayant la forme d'un escalier. Ils apportent un avantage au niveau de la compression du code; aussi, nous les traitons dans le paragraphe "factorisation" de ce chapitre.

3. CARACTERISATION DES CODES DE CONTOURS.

Une suite quelconque de 0,1,2,3 ne représente pas un contour. En cherchant les propriétés des codes de contours, nous visons une énumération possible pour les miniscontours, ainsi que la compression du codage; par exemple:

- une sous-suite constante peut être codée de manière plus efficace, ou ne pas être codée du tout si elle est présente dans tout contour;

- les délimiteurs entre deux types de codages (cf. plus loin: le problème de l'arrêt) peuvent être résolus par des sous-suites impossibles.

Soit C l'ensemble des suites de liens associées à un contour, et soit $S = l_1 \ldots l_n$ une suite quelconque de 0,1,2 ou 3. On note $\neg l$ le lien opposé à 1. Si S est associée à un contour, alors elle vérifie nécessairement:

(3.1) n est pair

(3.2) $l_1 = 2$, $l_n = 1$

(3.3) $l_{i+1} \neq \neg l_i$

Ces conditions ne caractérisent pas les séquences qui sont des contours, car il faut partir du point de mort et y revenir. Ainsi, la séquence 2 1 n'est pas un code de contour.

Soit S l'ensemble des suites qui vérifient (3.1), (3.2) et (3.3).

Proposition 3.1:

Pour n > 2 on a: $| S | < 3^{n-2}$

Preuve:

Soit $T = l_1, \ldots, l_{n-1}$ une suite de 0,1,2 ou 3 vérifiant:

(3.1), (3.3) et $l_1 = 2$

Il y a exactement 3^{n-2} suites de ce type; en effet:

$\forall j \in [2,n-1] \quad l_j \in \{ \{0,1,2,3\} - \neg l_{j-1} \}$

autrement dit: pour j > 1 tout l_j peut prendre 3 valeurs.

Ajoutons à chacune de ces suites un nième élément égal à 1. Les suites ainsi construites à partir de T sont toutes les suites possibles de longueur n vérifiant (3.1) et (3.2).

Il existe au moins une suite construite ne vérifiant plus la condition (3.3):

$T' = 2\ 3 \ldots 3\ 1$ (i.e. $\forall j \in [2,n-1]\ l_j = 3$)

□

Propriété:

Si $S \in \mathcal{S}$ est un code de contour alors il vérifie

(3.4) $\forall a \in \{0,1\}\ |\ \{l_i \mid l_i = a\}\ | = |\ \{l_i \mid l_i = \neg a\}\ |$

Preuve:

C'est l'idée triviale. En suivant le contour, nous sommes obligés de retomber sur le point de départ; il y a donc autant de liens montants que des liens descendants, et autant de liens allant à gauche qu'à droite.

□

Cependant nos contours ont une propriété supplémentaire qui n'est pas mentionnée dans la caractérisation précédente: ils sont donnés à partir de leur point de mort qui est le point le plus "en bas à droite" (avec priorité à "en bas"). Ceci impose des conditions supplémentaires sur les liens du contour.

Propriété:

Si $S \in \mathcal{S}$ est un code de contour, alors il vérifie

(3.5) $\forall i \in [1,n]\ \forall j$ tel que $1 \leqslant j \leqslant i$

$|\ \{l_j \mid l_j = 1\}\ | \leqslant |\ \{l_j \mid l_j = 3\}\ |$

On a bien sûr l'égalité pour $i = n$ d'après (3.4).

Preuve:

Si le nombre de liens descendants devient supérieur au nombre de liens montants, alors on contredit la définition du point de mort comme point "le plus bas". Cette condition assure donc qu'à tout moment on ne descend pas plus bas que le point de mort.

□

Remarque:

On pourrait croire que l'égalité dans la condition (3.5) n'est possible que lorsque i=n. Il n'en est rien comme le montre la figure suivante:

Figure 3.3
Exemple d'égalité dans (3.5) pour i<n

Par contre à droite du point de mort, il ne peut y avoir de points aussi bas que lui:

Propriété:

Si S ε $\mathcal{S}$ est un code de contour, alors il vérifie

(3.6) $\forall i \in [1,n]\ \forall j$ tel que $1 \leqslant j \leqslant i$

si $|\{l_j \mid l_j = 2\}| < |\{l_j \mid l_j = 0\}|$

alors $|\{l_j \mid l_j = 1\}| < |\{l_j \mid l_j = 3\}|$

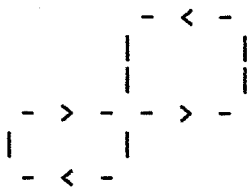
Preuve:

Chaque fois qu'on a l'égalité entre le nombre de liens montants et descendants, on est sur la même ligne que le point de mort. Dès lors on ne peut être à sa droite.

□

Ces propriétés ne sont pas suffisantes pour caractériser tous les codes de contours; l'expression du sens de parcours (connexité) manque, comme le montre la figure suivante:

Figure 3.4
Contre-exemple:



En parcourant dans le sens indiqué, on obtient le code
2 2 3 0 0 0 0 3 3 2 2 1 1 1
qui vérifie (3.1) à (3.6) et n'est pas un code de contour.

Notons S' l'ensemble des suites qui vérifient (3.1) à (3.6).

Jusque là on a: $|C| < |S'| < |S|$

Problème:

Evaluer ces valeurs.

Nous ne pouvons encore y répondre.

4. LE PROBLEME DE L'ARRET.

Il s'agit de trouver des séparateurs:

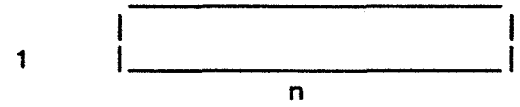
- entre deux codes de contours
- entre deux tronçons d'un même contour lorsque des codages différents sont proposés pour ces tronçons.

Ces séparateurs sont indispensables lors du décodage. De plus il faut pouvoir "annoncer la couleur", c'est-à-dire introduire un indicateur du type de codage lorsque des codages différents sont utilisés. Dans certains cas séparateur et indicateur peuvent être représentés par le même élément, par exemple lorsque la fin d'un codage implique forcément le début d'un autre codage connu.

Nous présenterons pour chaque type de codage le séparateur et l'indicateur éventuel associés. D'une manière générale, on peut mettre la longueur du codage devant, ou avoir une configuration impossible (par exemple deux liens opposés). Notons que **les configurations impossibles sont d'autant plus délicates (longues) que le codage est optimal.**

5. CODES OPTIMAUX DE SUITES DE LIENS.

Il est clair qu'une suite de liens 0,1,2,3 n'est pas un codage optimal d'un contour. Pour s'en rendre compte, il suffit de considérer le contour suivant:



Ce contour nécessite un code de (2n+2)2 bits.

Il est tout à fait naturel d'essayer de coder les 2n liens horizontaux d'un seul coup.

D'une manière générale, on peut voir un contour comme une chaîne de caractères et le problème devient celui de la factorisation de cette chaîne. L'alphabet considéré est ici {0,1,2,3} et nous pouvons utiliser toutes les techniques classiques de factorisation mises au point sur les mots.

Lorsque la factorisation donne des résultats médiocres, on peut utiliser des techniques tirant parti des configurations possibles dans les suites de liens.

5.1. FACTORISATION.

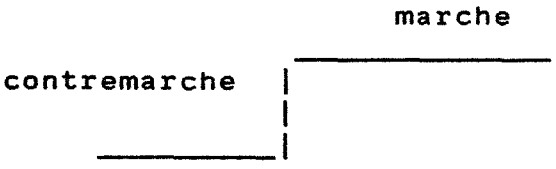
Nous avons implémenté une technique de factorisation qui revient à chercher des motifs (à deux éléments) que l'on peut associer à une vectorisation. Elle consiste à chercher des escaliers dans un contour. Cette idée est tirée d'une réflexion sur la reconnaissance de contours obtenus à l'aide de l'algorithme de tracé de vecteurs de BRESENHAM (cf. PAVLIDIS [PAVL]).

Le codage proposé est un codage exact.

5.1.1. Définition.

Un **escalier** sera défini par les types (sens et direction) de ses marches et contremarches, ainsi que par leurs tailles respectives.

Figure 3.5
escalier.



Les escaliers reconnus correspondent aux règles suivantes:

1. Les tailles des marches successives d'un même escalier sont à un près ($\pm$ 1), égales à celle de la première.

2. **Les contremarches sont toujours de taille 1.** C'est exactement ce qui se passe dans l'algorithme de tracé de vecteurs de BRESENHAM.

3. Par convention un escalier commence et se termine par une marche.

Remarque:

Un escalier peut avoir une seule marche.

5.1.2. Code d'un escalier.

code escalier::=(type escalier)(taille marche)
(codes des marches suivantes)

- 3 bits suffisent pour coder le type d'un escalier.
- théoriquement une marche peut atteindre la dimension maximale (hauteur ou largeur) de la page. Nous proposons un codage différentiel pour la taille de la marche, afin d'améliorer la performance du codage et de conserver l'indépendance par rapport à la taille totale de l'image. Il n'empêche que cette performance peut être améliorée par la connaissance de la taille de l'image.
- 2 bits suffisent à coder une marche.
- la première marche est implicite (de longueur taille_marche forcément).

Etudions en détail la signification de ces codes:

5.1.2.1. Type escalier.

type escalier	lien marche	lien contremarche
000	0	1
001	0	3
010	1	0
011	1	2
100	2	1
101	2	3
110	3	0
111	3	2

5.1.2.2. Code marche.

Soit l_0 la largeur de la première marche d'un escalier et l_i celle de la ième marche, nous avons alors:

code ième marche	largeur
00	$l_i = l_0$
01	$l_i = l_0 + 1$
10	$l_i = l_0 - 1$

Le code 11, inutilisé, va servir à indiquer la fin d'une suite de marches, et représente donc le code d'une marche finale fictive.

5.1.2.3. Taille marche.

Il faut faire une partition des tailles des marches en classes représentées par la longueur maximale possible dans chaque classe. Soient l_0 , l_1 ... ces bornes. La taille de la marche est alors codée par une entête représentant la classe, suivie de la taille sur:

$\log_2 (l_i - l_{i-1})$ bits.

Exemple:

taille marche	entête	code taille	soit au total
< 64	0	6 bits	7 bits
64≤ .. < 320	10	8	10
320≤ .. < 832	110	9	12
etc...	11..1	pour la dernière classe	

i.e. 320 = 64 + 256 donc code taille sur 8 bits, etc...

Remarques:

1. On ne peut utiliser une suite de 1 pour la dernière classe que si l'on connait la taille de la page. Sinon, le décodeur chercherait la longueur de cette entête sans pouvoir la trouver correctement. L'indépendance par rapport à l'échelle de saisie se payerait ici par un bit supplémentaire pour la dernière classe, ce qui n'est pas trop cher pour un escalier très long.

2. La partition en classes donne un codage d'autant plus performant que l'on a une idée précise de la fréquence des diverses tailles des marches. On pourrait alors adapter un code de Huffman pour les escaliers de chaque image séparément. Ceci se payerait par du temps de calcul avec plusieurs passes sur la même image d'une

part, et par le codage ...du code utilisé pour cette image d'autre part. On peut aussi trouver un moyen terme après l'analyse de quelques images (voir notre application).

5.1.3. Code mixte d'un contour.

Nous sommes maintenant en mesure de préciser de manière plus formelle le code de contour utilisé. Ce code mélange des codes de type liens (suite de liens tels qu'ils ont été définis pour les contours) et des codes de type escalier (comme définis ci-dessus).

```
code contour::=(séparateur)(code liens) |
               (séparateur)(code liens)(code escalier) |
               (séparateur)(code morceau)(code suite)

code morceau::=(code escalier) | (code liens)(code escalier)

code suite::=(séparateur)(code liens) |
            (séparateur)(code morceau) |
            (séparateur)(code morceau)(code suite)

code lien :::=(suite de liens terminée par un lien opposé
                                                au dernier)

code escalier::=(type escalier)(taille marche)(suite de codes
                de marches terminée par le code de la marche
                fin)
```

Commentaires sur ce code.

Le codage de début fonctionne en prenant pour séparateur la valeur booléenne suivante:
séparateur=1 ssi ce qui suit est un escalier.

(il nous suffit donc d'un bit pour coder ce séparateur).

En effet la suite de liens initiale d'un contour débute toujours par un lien de type 2 (bit gauche = 0). Lorsque le contour démarre directement par un escalier le premier bit du contour sera à 1 (séparateur); s'il commence par une suite de liens ce premier bit sera à 0.

La seule chose que nous n'avons pas encore précisée est la fin d'un tel code de contour.

Solutions envisageables:

1. coder en début de contour la longueur du contour.

2. terminer chaque contour par une situation impossible.

3. ne rien coder...puisque tout contour est fermé, et notre codage exact. En suivant le contour on doit retomber sur le point de départ, qui indique sa fin.

Si l'on envisage des déformations ultérieures sur les contours il faut choisir parmi les deux premières solutions. Celle qui nous parait la plus intéressante est celle d'un escalier impossible constitué d'une marche de taille nulle et ne contenant que la marche finale.

Ceci donne le code:
(séparateur)(type)(taille marche)(marche finale)
(1)(000)(0...0)(11)

donc un code de (6 + n) bits, où n est la longueur du code de la première classe des tailles des marches.

5.1.4. Algorithme de recherche des escaliers.

```
/* [intervalle] désigne [taillemarche-1 , taillemarche+1] */

Pour chaque contour

/* initialisations */

lire(lien)
marche:=lien;
comptmarche:=1;
ESCALIER:= FAUX;

tant qu'il existe un lien faire
lire(lien)
si (ESCALIER) alors
    si (lien=marche) alors
        comptmarche := comptmarche+1;
        si (comptmarche > taillemarche+1) alors
            terminer_escalier;
            ESCALIER := FAUX;
            comptmarche :=1;
        finsi
    sinon
        si (lien=contremarche) alors
            si (comptmarche >0) alors
                si (comptmarche ε [intervalle]) alors
                    coder_marche(comptmarche);
                    comptmarche:= 0;
                sinon /* la dernière marche vue est trop courte */
                    terminer_escalier;
                    coder_contremarche;
                    taillemarche:=comptmarche;
                    démarrer_escalier (marche,lien); /* même type */
                    comptmarche:=0;
                finsi
            sinon /* on a deux liens contremarche consécutifs */
                terminer_escalier;
                marche:=lien;
                comptmarche:=2;
                ESCALIER:=FAUX;
            finsi
        sinon /* le lien n'est pas une marche, ni une contremarche */
            si (comptmarche >0) alors
                si (comptmarcheε[intervalle]) alors
                    coder_marche(comptmarche);
                    terminer_escalier;
                    marche:=lien;
                    comptmarche:=1;
                    ESCALIER:=FAUX;
                sinon
                    terminer_escalier;
                    coder_contremarche;
                    taillemarche:=comptmarche;
                    démarrer_escalier(marche,lien);
                    comptmarche:=0;
                finsi
            sinon
```

```
                terminer_escalier;
                coder_contremarche;
                marche:=lien;
                comptmarche:=1;
                ESCALIER:=FAUX;
            finsi
        finsi
    finsi
sinon  /* il n'y a pas encore d'escalier */
    si (lien=marche) alors
        comptmarche:=comptmarche+1;
    sinon
        contremarche:=lien;
        taillemarche:=comptmarche;
        démarrer_escalier (marche,lien);
        comptmarche:=0;
        ESCALIER:=VRAI;
    finsi
finsi
fintant
```

5.1.5. Commentaires sur cet algorithme.

L'algorithme utilise les fonctions suivantes:

démarrer_escalier: écrit le séparateur nécessaire, code le type de l'escalier et la taille de la marche.

coder_marche: écrit le code de la marche en cours, avec comme paramètre la longueur de cette marche.

coder_contremarche: écrit le code lien de la dernière contremarche lorsqu'elle existe, car il a fallu arrêter un escalier (rappelons que par convention un escalier ne se termine jamais par une contremarche).

terminer_escalier: écrit le code fin de l'escalier en cours. Ce code n'est inséré dans le code de contour final que lorsqu'il est plus court que le code lien correspondant.

Cet algorithme de recherche des escaliers dans un contour est linéaire en fonction du nombre de liens du contour considéré. On procède donc en un seul balayage du contour, lien après lien.

La variable booléenne ESCALIER est vraie lorsqu'au cours de l'algorithme, l'escalier courant admet au moins une marche et une contremarche.

Par contre, tout au long de l'algorithme la variable marche contient la direction de l'escalier courant ou à considérer dès que l'on aura trouvé une contremarche.

Afin de régler le problème du dernier lien et du bout d'escalier associé (car cela ne tombe pas toujours juste!), nous avons choisi la solution suivante:

En fin de contour on force l'écriture de l'escalier en cours. Comme tous les escaliers, ce dernier n'est conservé que si sa longueur est plus courte que celle des liens correspondants.

5.2. A PROPOS DES MINIS-CONTOURS.

Le problème d'un codage optimal pour les minis-contours se pose. Cela revient à la connaissance structurelle de tous ces contours. A la condition que leur nombre ne soit pas trop grand, le codage par le numéro du contour deviendrait très efficace (i.e. nombre de bits pour coder un numéro par rapport au nombre nécessaire pour la suite des liens).

Nous poursuivons cette étude de recherche d'un compromis entre:

- une table de tous les contours, efficace mais difficile à coder,
- et une description algorithmique, moins efficace mais facile à coder.

Nous présentons ci-après les premiers contours, classés pour chaque périmètre suivant les surfaces décroissantes et en juxtaposant les contours symétriques.

*C*i désigne l'ensemble des contours de longueur i.

*C*4 = 1

```
X
```

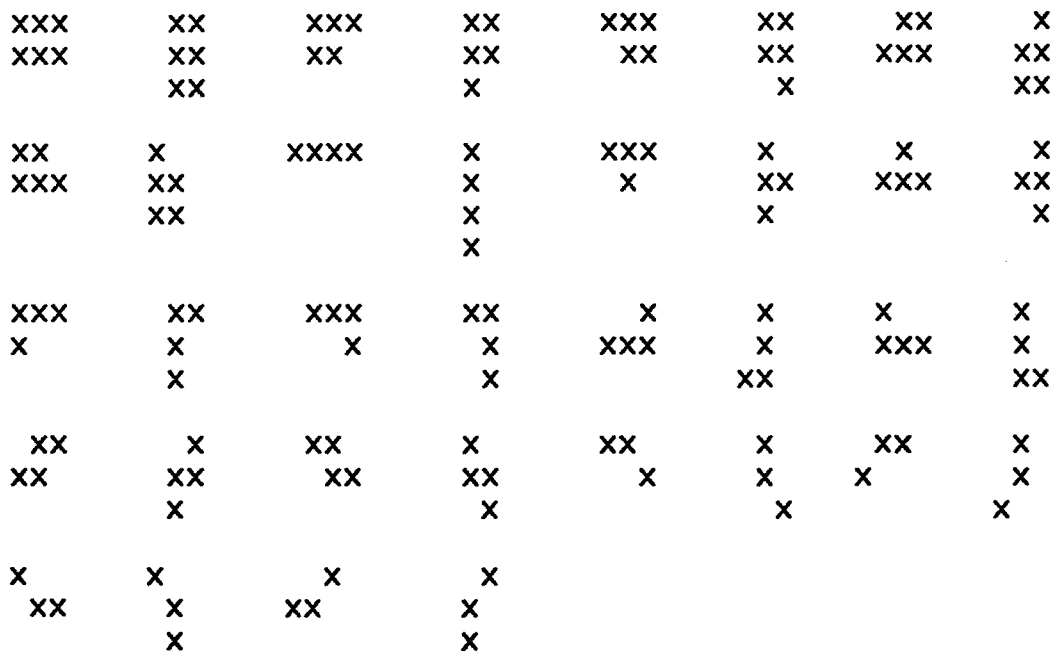
*C*6 = 2

```
XX     X
       X
```

*C*8 = 9

```
XX     XXXX     X     XX     XX     X     X     X     X
XX              X     X       X    XX    XX      X   X
                X
                X
```

$C_{10} = 36$



Soit jusqu'ici 48 contours pour les longueurs inférieures ou égales à 10, donc 6 bits suffiraient, ce qui est plus intéressant que les codages que nous proposons ci-après.

Ainsi, connaître l'ensemble des minis-contours ayant n liens, soit de manière exhaustive à l'aide d'un procédé énumératif, soit de manière plus structurelle (un algorithme "simple" capable de reconstruire un code de contour à partir de son numéro) semble permettre une amélioration du taux de compression.

Cependant, cette voie de recherche paraît délicate si l'on se réfère à REDELMEIER [REDE]. En effet, on retrouve de tels objets, minis-contours, sous le nom de **polyominos**, (ou encore "animaux") dans la littérature. Bien que les polyominos aient été étudiés depuis longtemps (cf. GOLOMB [GOL1]), leur étude combinatoire en vue des applications au traitement des images reste à faire (cf. BERGE, CHEN, CHVATAL et SEOW [BERG]).

5.3. CODAGES DIFFERENTIELS.

Nous nous intéressons maintenant au codage d'une suite de liens sur laquelle une factorisation de type "escalier" donne des résultats médiocres. Ceci concerne typiquement les contours des caractères, plus généralement les petits contours, où les changements de direction sont fréquents, ainsi que les suites de liens entre deux escaliers. L'idée que nous exploitons est qu'il n'est pas toujours nécessaire de coder un lien sur 2 bits.

5.3.1. Codage différentiel brut.

Puisqu'un lien 1 ne peut être suivi de son opposé, les liens suivants possibles sont:

- l
- l+1 (modulo 4)
- l-1 (modulo 4)

ce qui donne une première idée de codage:

0 lorsqu'un lien est identique au précedent,
10 pour l+1,
11 pour l-1.

Exemple:

la suite 2 2 3 3 3 0 3 2 2 1 1 sera codée
(2) 0 10 0 0 10 11 11 0 11 0

soit 15 bits au lieu de 20.

Remarque:

Nous ne comptons pas le premier lien puisque tout contour commence par un lien 2. Donc il n'y a pas besoin de le coder.

Ce codage permet de gagner un bit pour chaque lien égal au précédent, et surtout **ne provoque pas de pertes.** Par contre:

- il n'exploite pas la moindre possibilité de factorisation,

- Le seul arrêt possible est celui du retour au point initial du contour, puisqu'il est constitué d'une suite tout à fait quelconque de bits.

5.3.2. Essai de factorisation.

On ne peut espérer un résultat positif d'une factorisation qu'à la condition d'avoir des pertes minimes pour une suite de liens telle que chaque lien est différent de son suivant immédiat; plus généralement, il faudra minimiser les pertes pour les suites très courtes de liens identiques. Or l'exigence minimale d'une factorisation de type "escalier" est de 6 bits (i.e. séparateur + type escalier + fin), plus la longueur du code de la taille de l'escalier. Les pertes occasionnées sont alors trop importantes.

En regroupant les liens identiques, un seul bit suffit pour coder le type de lien, puisque seuls subsistent les cas lien + 1 et lien - 1. Nous proposons le codage suivant:

nombre de liens identiques	entête	code longueur	longueur totale du code
⩽1 .. < 3	0	1 bit	3 bits
⩽3 .. < 5	10	1	4
⩽5 .. < 7	110	1	5
⩽7 .. <11	1110	2 bits	7
⩽11 .. <19	11110	3	9

etc ...

Ce codage minimise les pertes lorsque le nombre de liens identiques est faible. Les gains sont par conséquent faibles, mais commencent à partir de 4 liens identiques par rapport au codage différentiel brut (qui code sur n + 1 bits n liens identiques), comme l'indique le tableau suivant:

nombre de liens	incidence
1	perte 1 bit
2	=
3	=
4	gain 1 bit
5	" 1
6	" 2
7	" 1
8	" 2
9	" 3
10	" 4
11	" 2
....	

Le problème de l'arrêt peut être résolu ici avec une entête particulière: la suite de 1 la plus longue possible malheureusement, puisque les longueurs nulles sont déjà utilisées. Mais:

- le principe de l'arrêt sur le point initial du contour n'est jamais remis en cause,

- on peut cette fois ci l'appliquer sur une suite de liens entre deux escaliers, car une telle suite comporte forcément un nombre de liens identiques consécutifs faible - elle aurait fait l'objet d'un escalier

sinon (cf. chapitre "application").

5.4. CODAGE PAR PAQUETS DE LIENS.

Considérons une séquence S de k liens: l_1 ,...., l_k.

Soit $l_0 \in \{0,1,2,3\}$ le lien qui précède cette séquence.

Nous avons déjà remarqué dans les codages différentiels que

$\forall i\ 1 \leq i \leq k\ l_i \in \{0,1,2,3\} - \neg l_{i-1}$

Pour coder S il suffit de se souvenir des différences successives par rapport à l_0 et ceci peut se faire par additions successives modulo 4 des valeurs {0,1,3}.

Il y a donc 3^k séquences différentes de k liens.

L'idée de ce code consiste à diviser un code de contour en paquets de k liens. A chaque paquet on associe un numéro et à l'inverse, au décodage, nous saurons trouver la suite de k liens associée (soit algorithmiquement, soit par tableau). Le problème de l'arrêt peut être reglé de façons différentes:

- par des numéros particuliers, indiquant que le contour s'arrête après le premier (resp. ième) lien du paquet suivant (ou précédent),

- par le retour au point de mort,

la première solution restant valable pour les codages non exacts, alors que la deuxième ne peut être retenue que pour les codages exacts. L'initialisation se fait sans problème puisque le premier lien est toujours 2.

Il s'agit de fixer k. Ceci doit être un compromis entre le gain obtenu et le temps du décodage. En outre k ne peut être trop grand, sinon le dernier paquet non rempli risque de coûter cher.

Exemple:

Pour k = 5 on a 3^5 = 243 séquences possibles, et on peut coder ainsi sur 8 bits (2^8 = 256 donc > 243) une séquence de 5 liens. Il reste 13 valeurs disponibles avec lesquelles on codera:

v_1 lorsqu'on s'arrête après le premier lien
.
.
v_5 cinquième lien.

Moralité:

A partir de ces codages on peut construire une infinité de variantes mélangeant les avantages et inconvénients des uns et des autres, et nous laissons le lecteur libre de s'en construire un ...

CHAPITRE 4

UNE APPLICATION COMPRESSION DE DOCUMENTS TECHNIQUES

1. PRESENTATION.

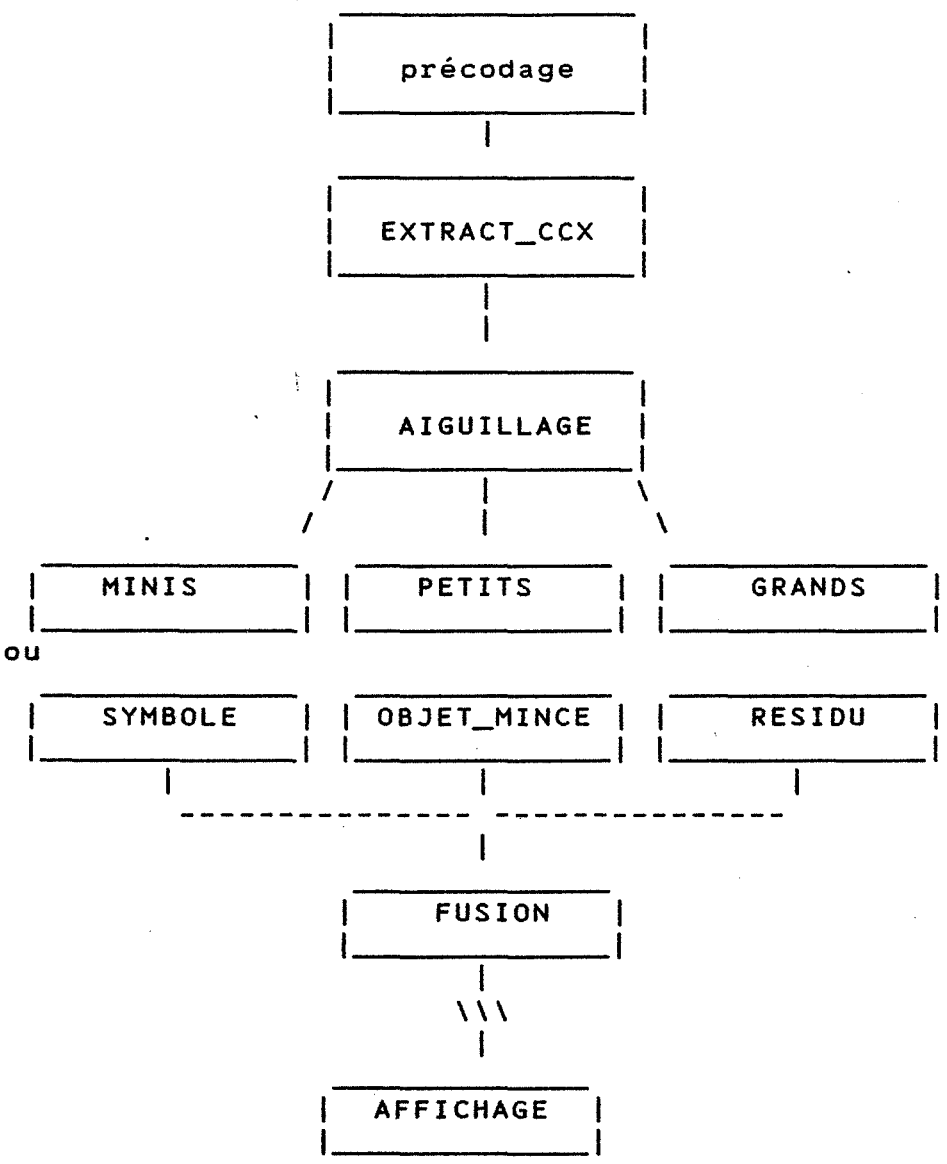
Dans le cadre du projet SARDE de télématisation de la documentation technique des télécommunications [JOLY, ROM1], nous avons réalisé une version permettant de valider la méthode. Le projet SARDE comprend des études sur la saisie (scanner), la compression, le stockage (Bases de données sur disque optique numérique) et la restitution (écran haute résolution) de documents manuscrits et/ou dactylographiés et de schémas techniques en noir et blanc. Les parties qui nous intéressent ici portent sur les phases de:

- saisie, à partir de la binarisation par le scanner, donc après la phase de transformation des niveaux de gris en noir/blanc,
- compression
- restitution locale: le vecteur de transmission à distance n'entre pas en jeu ici.

1.1. DECOUPAGE EN MODULES.

Nous présentons les spécifications de notre logiciel ainsi qu'une programmation monoprocesseur, réalisée sous UNIX (système SMX) sur une machine SM90 (CNET). Dans ces spécifications il a été largement tenu compte d'une parallélisation qui interviendra ultérieurement. La figure suivante présente de façon sommaire, la décomposition du traitement en modules et l'interconnexion de ceux-ci.

Figure 4.1
Principaux modules.



Le premier module a pour tâche de balayer l'image et de fournir en sortie **le précode de l'image.** Nous justifierons l'intégration de ce module dans le matériel de saisie.

le deuxième module est celui de détection des composantes connexes. A ce niveau, une composante connexe est représentée par son contour extérieur et ses contours intérieurs. On lui associe également les attributs mesurés au fur et à mesure du balayage.

Le module d'aiguillage analyse les attributs de chaque composante connexe qui lui est transmise et détermine comment doit être traité l'objet.

On trouve ensuite les modules spécialisés dans le codage de chacune des classes d'objets. Nous donnons quelques résultats en ce qui concerne la séparation en symboles, objets minces et résidus, mais nos efforts dans la suite ont porté plus sur l'étude des petits et grands contours.

Le module fusion se charge de recueillir les codes des composantes connexes ou contours de différents types afin de transmettre au module d'affichage le code complet d'une image.

Le module affichage quant à lui se charge de la reconstruction (décodage) de l'image codée précédemment et de l'afficher sur une sortie Bitmap. Ce module pourra naturellement être localisé sur d'autres sites (autant qu'il y aura de sites de consultation des documents numérisés). Il doit donc être indépendant.

Le découpage de l'application en modules n'est pas arbitraire. Il doit permettre, pour chaque module, un déroulement indépendant de celui des autres. Ceci est indispensable pour envisager une parallélisation de l'application. Dans ce cas un dernier module devrait donc récupérer les codes fournis dans un ordre imprévisible (asynchronisme) par les divers modules de codage et construire le code global de l'image. La version actuelle doit être considérée comme un version **"pipeline"** dans laquelle chaque module est représenté par un processus. De plus il apparaît que les modules d'aiguillage et de fusion ne peuvent être dupliqués dans l'optique choisie ici. Il reste donc à s'assurer que ces tâches ne constituent pas le goulot d'étranglement de l'application en ce qui concerne le temps de traitement.

1.2. CONVENTIONS D'IMPLEMENTATION.

Nous avons choisi de réaliser le logiciel en langage C. Il est évident que si ce logiciel est amené à être implémenté sur une architecture spécialisée, les performances que nous indiquons ne peuvent qu'être améliorées...

L'affichage a été essayé sur les bitmaps ORION (MYFRA), qui permet de visualiser une page A4 entièrement(2288 lignes x 1728 colonnes), et EHR90 (NUMELEC) qui possède une résolution bien plus faible: 780 lignes x 1024 colonnes.

1.2.1. Taille des documents.

Les documents utilisés lors des essais (fournis par le CNET) sont de taille A4 et ont été numérisés avec un facteur de résolution de 8 points/mm. Leur taille est de 2288 lignes x 1728 colonnes (soit environ 4 Mégapixels). Ceux du CCITT (également fournis par le CNET) sont de dimension 2382 lignes x 1728 colonnes.

1.2.2. Allocation dynamique de mémoire.

Lors des créations des structures de données associées aux composantes connexes, nous avons utilisé les fonctions d'allocation du système UNIX : **malloc, realloc** et **calloc.**

Comme ces structures peuvent être énormes (jusqu'à 1 Méga octet pour une image A4) nous avons systématiquement restitué l'espace mémoire dès que possible à l'aide de la fonction système **free.**

1.2.3. Organisation du développement des programmes.

Dans un premier temps, nous construisons une maquette de validation des principes choisis. Cette maquette est constituée de modules exécutables indépendants.

Les liaisons entre modules exécutables sont réalisées sous forme de "pipes". En effet, le principe des pipes est suffisamment simple pour que l'on puisse envisager de l'implémenter sur un système multiprocesseur.

Cette technique introduit toutefois l'"overhead" suivant: la communication d'une composante connexe est réalisée par sa transmission explicite et intégrale sur le pipe de liaison, alors que dans une application formée d'un seul module exécutable composé de plusieurs processus, ceux-ci pourraient se communiquer des composantes connexes, en transmettant seulement sur le "pipe", leurs adresses en mémoire commune.

2. PRECODAGE.

2.1. LES BORDS DU PRECODE.

Comme les codes de contours manipulent des interpixels, nous avons adopté la convention suivante:
L'image est bordée par un cadre de pixels blancs.

Ces derniers pixels sont virtuels et donc ne sont pas affichés, mais par contre ils permettent un traitement homogène des changements de couleur aux bords de l'image.

2.2. CALCUL DU PRECODE.

Le temps de ce module est pratiquement constant d'une image à l'autre et est actuellement de l'ordre de 9mn30s par image. Ce temps dépasse largement celui de tous les autres modules, mais il faut prendre en compte les remarques suivantes:

- Les images que nous avons traitées sont, au départ, codées par plages, et nous avons transformé ces plages en flots de bits. Ainsi tout se passe comme si le scanner délivrait l'image numérisée sous forme d'un fichier Unix, et le module précode prend alors ce

fichier en entrée.

- Ce module doit traiter chacun des 4 Mégapixels, pour calculer le précode de chaque interpixel. Dans un langage évolué, ce traitement représente une centaine d'instructions élémentaires. En prenant comme temps d'exécution moyen d'une instruction sur notre machine 1.5 µs, on arrive au temps que nous avons.

Ceci confirme notre analyse (voir chapitre 2), et il nous semble que la simplicité des opérations à effectuer permet, sinon impose, la réalisation d'une carte spécialisée à insérer à la sortie du scanner. Une programmation en assembleur doit déjà permettre une division de ce temps par 4 à 5.

Nous donnons ci-après les tailles (en octets) des précodes des images, comparé au codage par plages. Le codage par plages est unidimensionnel, où chaque ligne est codée sur 16 bits pour le numéro de ligne, suivi d'une suite de doublets de 16 bits correspondants aux débuts et fins des plages noires dans la ligne. Le précode se compose de 4 bits par configuration utile (précodes 1 à 14). Les coordonnées des points de mort sont codées avec 16 bits chacun, sur un mot complet, le mot incomplet précédent reste inoccupé.

image	code plages	précode	image	code plages	précode
cnet1	50236	65024	ccitt1	88054	101376
cnet2	75390	57344	ccitt3	171972	164352
cnet3	206682	313856	ccitt5	174364	181760
cnet4	152862	163840	ccitt6	105510	115200
cnet5	123728	151040	ccitt7	278512	366080
cnet6	112266	128000			
cnet7	103418	110592			
cnet8	351092	247808			
cnet9	96900	110592			
cnet10	181330	238592			
cnet11	241666	237056			
cnet12	114386	139776			
cnet13	151708	194048			
cnet14	152962	197632			

3. EXTRACTION DES COMPOSANTES CONNEXES.

3.1. RESULTATS.

Nous ne cherchons pas encore ici la compression maximale, puisque chaque composante connexe contient un nombre d'informations (attributs) non négligeable destinées à son traitement ultérieur. A la sortie de ce module chaque composante connexe est décrite par:

- nombre de ses contours, sur 16 bits,
- hauteur et largeur, sur 16 bits chacune,
- surface noire, sur 32 bits,

suivie de la description de chaque contour par:

- longueur du contour, sur 16 bits,
- coordonnées ligne et colonne, sur 16 bits chacune,
- surface du contour, sur 32 bits,
- codage des liens, avec 2 bits par lien.

La plus petite composante connexe possible (1 pixel noir) occupe à ce stade 168 bits. Les tailles des images sont données en octets dans les résultats suivants:

image	taille	temps d'exécution	image	taille	temps d'exécution
cnet1	28508	34s	ccitt1	45311	1mn05s
cnet2	27455	34s	ccitt3	71207	1mn41s
cnet3	105380	4mn06s	ccitt5	86884	1mn51s
cnet4	62460	2mn08s	ccitt6	37315	2mn57s
cnet5	66474	2mn35s	ccitt7	190957	3mn17s
cnet6	70026	2mn18s			
cnet7	59207	1mn48s			
cnet8	96529	13mn09s			
cnet9	54461	1mn02s			
cnet10	154400	3mn21s			
cnet11	149358	4mn05s			
cnet12	60473	1mn45s			
cnet13	86777	1mn58s			
cnet14	84843	2mn07s			

3.2. COMMENTAIRES.

On peut remarquer à travers ces résultats que le temps de l'extraction des composantes connexes ne dépend pas seulement du nombre d'objets à extraire, mais aussi de la taille des parois nécessaires à la construction des contours. En effet:

- pour des images de même type (par exemple, cnet 1 12 13 et 14 sont des textes faits sur machine à écrire) les objets sont semblables de par leur taille, et le temps semble alors proportionnel au nombre d'objets extraits.

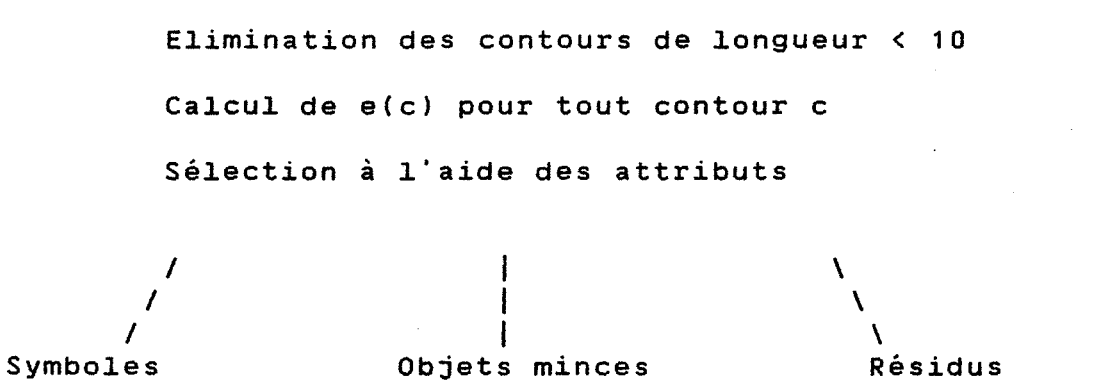
- la gestion des listes de liens d'une paroi (cf. chapitre 2, "gestion des listes de liens"), provoque beaucoup de réallocations de mémoire pour une paroi très longue, et chaque réallocation peut engendrer une

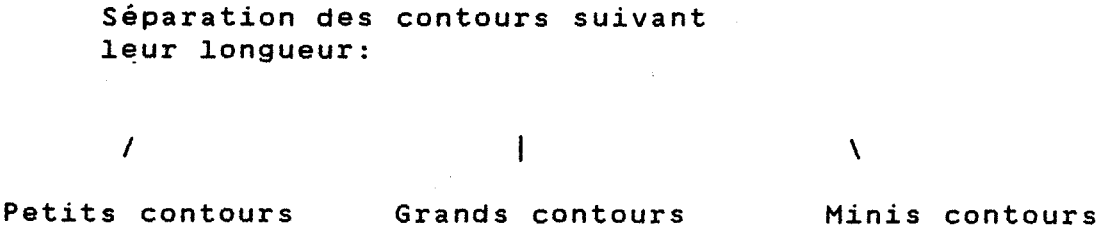
recopie de l'espace précédent. Nous avons pris des incréments d'un mot (32 bits dans notre cas, donc 32 liens de parois). Il apparait ici qu'une réallocation dépendant de la taille déjà allouée peut être intéressante. L'image cnet8 est caractéristique à cet égard: une seule composante connexe occupe plus de la moitié de l'image (voir la taille et le nombre d'objets dans le paragraphe suivant).

4. *MODULE D'AIGUILLAGE.*

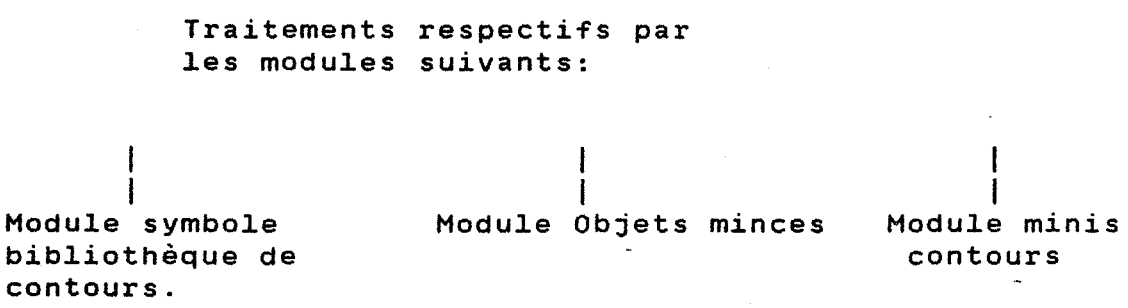
4.1. ORGANIGRAMME GENERAL.

On note e(c) l'application qui à un contour c associe le contour obtenu à l'issue de l'algorithme d'élimination des barbules. Ce module se présente comme suit:





Calcul de $e^2(c)$
pour tout contour c.



4.2. REPARTITION DES COMPOSANTES CONNEXES.

Un symbole est contenu dans une boite 32x32. Si cette mise en boite échoue, la composante connexe est un objet mince lorsque

surface noire / perimètre < 2 + ε,

sinon, c'est un résidu.

En jouant sur ε dans la plage $0<\varepsilon<1$, on fait passer de l'état de résidu à l'état d'objet mince les composantes connexes formées par 2 ou 3 caractères qui se touchent par exemple. Avec $\varepsilon \backsim 1$, on obtient des résidus conformes à l'idée qu'on s'en fait, à savoir des objets dont la description du contour est très longue (jusqu'à plusieurs dizaines de milliers de liens), contenant éventuellement beaucoup de petits contours intérieurs. Ceci est caractéristique des feuilles quadrillées dans lesquelles les caractères et le quadrillage sont connectés. Les résultats présentés ont été obtenus avec $\varepsilon = 0.9$

La tableau suivant donne le nombre de composantes connexes par type ainsi que leur taille avec les mêmes conventions indiquées lors de l'extraction de ces composantes: 80 bits pour l'entête de chaque composante connexe, et 80 bits pour l'entête de chaque contour.

4.2.1. Nombre et taille (en octets) des composantes connexes.

Sans élimination des barbules.

Dans la colonne "éliminés" on a le nombre de composantes connexes, puis le nombre de contours intérieurs.

image	symboles	objets minces	résidus	éliminés	
cnet1	508	37	0	39	0
	23946	3504	0		
cnet2	144	30	3	66	13
	6018	2282	17453		
cnet3	609	58	3	415	55
	20946	4972	69430		
cnet4	543	80	3	246	31
	21387	30899	4243		
cnet5	1087	29	19	291	249
	37137	4633	15164		
cnet6	838	78	0	1048	210
	28406	16293	0		
cnet7	798	63	0	767	72
	29185	12371	0		
cnet8	809	64	1	268	25
	31806	9757	48438		
cnet9	809	142	1	485	0
	29839	13509	213		
cnet10	1923	212	1	2421	3
	57534	15684	27679		
cnet11	1284	109	4	2438	17
	39394	9696	46670		
cnet12	1107	41	0	131	7
	52483	4649	0		
cnet13	1671	31	1	170	33
	78457	3062	609		
cnet14	1656	42	0	46	5
	78366	4817	0		

image	symboles	objets minces	résidus	éliminés	
ccitt1	1007	24	4	10	15
	42153	1977	455		
ccitt3	1344	45	1	18	127
	56366	12338	136		
ccitt5	1919	21	2	96	220
	72806	6673	2038		
ccitt6	443	28	6	20	75
	16943	2106	16753		
ccitt7	4546	7	0	1395	298
	155031	655	0		

4.2.2. Elimination des barbules.

Nous donnons ci-après la taille des **codes des contours seuls** pour chaque type d'objet avant et après l'élimination des barbules.

image	symboles		objets minces		résidus	
	avant	après	avant	après	avant	après
cnet1	11986	11022	2444	2230		
cnet2	2658	2478	1662	1605	16463	16193
cnet3	8196	7342	3662	3278	64180	58835
cnet4	9357	8472	28529	26379	3993	3741
cnet5	14687	13758	3953	3722	14074	13132
cnet6	10896	9482	14273	12889		
cnet7	12135	10816	11051	9957		
cnet8	13606	12562	8147	7497	44808	42424
cnet9	12379	11451	10129	9622	173	169
cnet10	18044	16027	11224	10344	26779	24850
cnet11	12654	11317	7266	6862	42730	40258
cnet12	26013	23358	3249	2932		
cnet13	38217	34294	2032	1795	549	488
cnet14	38586	34688	3407	3110		
ccitt1	18793	17088	1487	1409	335	317
ccitt3	25666	23762	11358	11177	66	58
ccitt5	30116	27331	6233	6126	1788	1754
ccitt6	6973	6381	1496	1432	15813	15420
ccitt7	59081	50867	425	402		

4.3. REPARTITION SELON LA LONGUEUR DES CONTOURS.

Le problème essentiel ici est le choix de la valeur séparant petits et grands contours. Nous avons cherché la plage de valeurs "la plus neutre possible", c'est-à-dire celle correspondant au nombre le plus faible d'objets passant d'une catégorie à l'autre. Les résultats sont donnés pour une séparation à 200.

Le module travaillant sur les minis contours n'a pas encore été implémenté. Dans l'état actuel ces contours font partie des petits contours avec un traitement identique.

Les contours éliminés sont les mêmes. Leur nombre est donc égal au total des composantes connexes et contours intérieurs trouvés dans le cas précédent.

Les traitements à effectuer dans la suite étant connus pour ces objets, il ne faut conserver dans les sorties que les données nécessaires. Aussi, pour connaitre les performances sur les codes de contour, il faut prendre en compte dans la taille des objets:

- pour les petits contours,
  - * les coordonnées de chaque contour, sur 2x12 bits pour l'instant
  - * la longueur du contour sur 10 bits (essentiellement pour des raisons d'essais de longueurs différentes, avant de la fixer à 8 bits dans notre cas)
  - * la surface sur 16 bits
- et pour les grands contours,
  - * les coordonnées, sur 2x12 bits
  - * la longueur sur 16 bits

4.3.1. Nombre et taille (en octets) des classes de contours.

Avec élimination des barbules (e(c)), puis $e^2(c)$ pour les grands contours.

image	petits contours nombre	petits contours taille	grands contours nombre	grands contours taille	grands après $e^2(c)$
cnet1	727	16116	30	1832	1796
cnet2	257	6284	63	15916	15888
cnet3	960	17512	301	59424	59200
cnet4	739	15268	100	28420	28336
cnet5	1228	22240	59	16324	16280
cnet6	994	17244	43	11556	11492
cnet7	942	17444	34	9376	9320
cnet8	1276	24480	194	46876	46532
cnet9	1084	22164	52	6092	6032
cnet10	2235	34676	114	31052	30584
cnet11	1724	31668	190	38436	38008
cnet12	1619	34612	20	1888	1860
cnet13	2417	50420	13	1312	1292
cnet14	2408	51132	13	1764	1748
ccitt1	1344	25900	18	1364	1328
ccitt3	1742	34696	43	11396	11384
ccitt5	2379	42688	17	7384	7368
ccitt6	621	11304	54	16064	15992
ccitt7	5054	81980	11	600	560

4.4. COMMENTAIRES.

L'élimination des barbules réduit la taille des contours et n'introduit pas de défauts visibles, du moins sur les documents que nous avons utilisés. En effet, après affichage des diverses classes avec et sans élimination de ces barbules, nous n'avons pas trouvé de défauts. Sur certains documents, la qualité s'en trouve améliorée, du fait aussi de l'élimination des contours de longueur inférieure à 10.

Mais l'effet de cette opération se fait sentir surtout lors des étapes suivantes, en particulier lors de la recherche des escaliers.

5. BIBLIOTHEQUE DE PETITS CONTOURS.

Nous avons réalisé une bibliothèque des petits contours (dont la longueur est inférieure à 200). L'hypothèse qui sous-tend ce travail: la longueur, la surface et une suite de points particuliers définissent un contour et nous pourrons donc les comparer au moyen de ceux-ci.

5.1. STRUCTURE DE LA BIBLIOTHEQUE.

Pour chaque longueur, un ensemble d'entêtes de bibliothèque caractérise les contours de bibliothèque de même longueur.

Une entête désigne l'ensemble des éléments de bibliothèque ayant même nombre de points caractéristiques.

Un élément de bibliothèque contient la surface du contour, la description du contour par ses liens, les points caractéristiques, et les coordonnées des contours semblables.

Figure 4.2
Structure de la bibliothèque.

La structure de la bibliothèque se décompose ainsi:

- Un tableau de pointeurs sur une première entête. Pour chaque longueur de contour on a donc un élément de tableau (pentête).

- Pour chaque "pentête" une suite d'entêtes contenant le nombre de points caractéristiques, un pointeur sur le premier élément de bibliothèque pour ce nombre de points (pelbib) et un pointeur sur l'entête suivante - pour la même longuer de contour -.

- L'ensemble des éléments de bibliothèque pointés par une même entête est géré aussi comme une liste chaînée contenant pour chaque élément la surface et les pointeurs sur:

  * le code des liens du contour

  * la suite des points caracteristiques

  * la suite des contours semblables

  * l'élément de bibliothèque suivant ayant le même nombre de points caractéristiques.

- Les points caractéristiques ainsi que les coordonnées des contours semblables forment des listes chaînées.

5.2. GESTION DE LA BIBLIOTHEQUE.

L'accès à la bibliothèque se fait de la façon suivante:

Pour chaque contour C de longueur L et de surface S, on procède à l'extraction de ses points caractéristiques (définis au paragraphe suivant).

On cherche s'il existe dans la bibliothèque un contour C' ayant:

- une même longueur à ε(longueur) près.

- **exactement** le même nombre de points caractéristiques.

- une même surface à ε(surface) près.

Lorsqu'un tel contour C' existe, on procède alors à la comparaison de leurs suite de points caractéristiques.

Pour ce faire on vérifie successivement si les coordonnées des points caractéristiques de deux contours se correspondent à ε(lignes) et ε(colonnes) près (calcul non cumulé d'un point à un autre).

Lorsque C' n'est pas trouvé, C devient un nouvel élément de bibliothèque.

Pour une classe de contours semblables, l'élément de bibliothèque est alors le premier contour rencontré dans l'ordre lexicographique. Ceci peut provoquer une création de nouveaux éléments de bibliothèque qui seraient semblables selon nos critères à l'un des contours de la classe autre que l'élément de bibliothèque. Or le temps nécessaire à la comparaison de tous les éléments d'une classe, et la conservation de tous les points caractéristiques, interdisent cette action. Nous avons essayé une autre méthode permettant de constituer des **classes d'équivalence** de la façon suivante:

Lorsque le contour C' est trouvé, on cherche s'il existe un deuxième contour C'' toujours parmi les éléments de bibliothèque et selon les mêmes critères de comparaison, semblable à C. Si C'' est trouvé, alors les deux classes sont fusionnées, et C devient le nouveau représentant de la nouvelle classe. Il faut dans ce cas conserver toute la structure de la bibliothèque jusqu'à la fin du traitement de l'image, et ceci justifie la liste des semblables dans la structure, ainsi que le pointeur sur la suite des liens de chaque élément de bibliothèque.

5.3. POINTS CARACTERISTIQUES.

Il y a quatre directions possibles pour un contour:

en haut et à gauche (suite de liens 2 et 3).

en haut et à droite (suite de liens 3 et 0).

en bas et à gauche (suite de liens 1 et 2).

en bas et à droite (suite de liens 1 et 0).

Définition:

Un point de contour est dit **caractéristique** s'il est associé à un changement de direction non négligeable.

En un seul balayage du contour nous construisons la liste des points dits **caractéristiques** d'un contour.

Le point de mort d'un contour est le premier point caractéristique, ensuite à chaque changement de direction du contour, nous introduisons dans la liste, un nouveau point caractéristique, sous réserve qu'il soit à une distance ε(pointcarac) du précédent.

5.4. PARAMETRES DE CONTROLE DE LA BIBLIOTHEQUE.

Il s'agit de trouver un bon compromis entre la fidélité de la restitution et le taux de compression.

Il y a 5 paramètres majeurs qui régissent cette bibliothèque.

ε(longueur)

ε(surface)

ε(ligne)

ε(colonne)

ε(pointcarac)

Les paramètres ε(longueur), ε(surface) ainsi que le nombre de points caractéristiques représentent le filtre avant la comparaison des points caractéristiques des deux contours.

Les paramètres ε(ligne) et ε(colonne) sont actuellement égaux. Ils représentent l'écart maximal admis lorsqu'on compare 2 points caractéristiques; il n'y a pas de cumul des écarts successifs: dès qu'un point est en dehors de ces limites la comparaison s'arrête avec un résultat négatif.

ε(pointcarac) est utilisé lors de la détection des points caractéristiques. Il représente la distance minimale entre deux points caractéristiques (en ligne ou colonne).

Pour l'instant nous avons privilégié la fidélité de restitution par rapport au taux de compression. Les paramètres sont modulés en fonction de la longueur des contours, et sont d'autant plus strictes (petits) que la longueur est petite.

5.5. RESULTATS.

Dans un premier temps nous avons codé l'ensemble des petits contours sous la forme: élément de bibliothèque suivi de tous ses contours semblables. Nous utilisons un bit pour indiquer si les coordonnées sont suivies du code de contour (c'est donc un élément de bibliothèque), ou si c'est un élément semblable au dernier de la bibliothèque. Les contours sont codés selon le code différentiel non factorisé. On a donc une suite de:

- ligne, 12 bits
- colonne, 11 bits
- indicateur élément de bibliothèque ou contour semblable, 1 bit
- code contour lorsque c'est un élément de bibliothèque

image	taille	nombre de contours / biblio.	nombre de contours \ au départ	temps	taille coordonnées
cnet1	6408	318	727	16s	2090
cnet2	3028	184	257	5s	738
cnet3	8988	562	960	23s	2760
cnet4	8376	529	739	18s	2124
cnet5	8540	476	1228	24s	3530
cnet6	9376	568	994	23s	2857
cnet7	8732	521	942	21s	2708
cnet8	9788	515	1276	30s	3668
cnet9	9420	528	1084	24s	3116
cnet10	14468	669	2235	54s	6425
cnet11	12728	613	1724	41s	4956
cnet12	10048	400	1619	42s	4654
cnet13	13616	509	2417	1mn09s	6949
cnet14	14108	520	2408	1mn11s	6923
ccitt1	8820	435	1344	30s	3864
ccitt3	12632	625	1742	50s	5008
ccitt5	13164	614	2379	1mn	6839
ccitt6	5632	379	621	12s	1785
ccitt7	33324	1682	5054	4mn03s	14530

On trouvera en annexe la répartition des contours selon les éléments de bibliothèque et leur longueur.

5.6. COMMENTAIRES.

Un essai de factorisation des codes de contour n'a pas donné des résultats permettant de conclure, et dans l'ensemble les pertes et gains obtenus étaient faibles.

Les classes d'équivalence donnent un gain jusqu'à 7% sur la taille totale des contours.

Pour ces contours il serait intéressant d'obtenir un gain sur le codage des coordonnées.

6. TRAITEMENT DES GRANDS CONTOURS.

6.1. DETECTION DES ESCALIERS.

Nous appliquons à ces contours l'algorithme de détection d'escaliers, avec un codage différentiel des tailles des marches. Dans un premier temps les liens entre deux escaliers restent codés sur 2 bits chacun, et les coordonnées sur 23 bits. Les coordonnées ne tiennent pas une place importante, vu le nombre faible de contours dans ce cas. Nous obtenons les premiers résultats sur les taux de compression avec:

- 494208 / (taille de l'image comprimée) pour les images cnet,

- 514512 / (taille de l'image comprimée) pour les images ccitt.

image	grands contours / avant	\ après	temps	total image	taux compression
cnet1	1796	1480	1s	7888	62.7
cnet2	15888	2212	6s	5240	94.3
cnet3	59200	30960	34s	39948	12.4
cnet4	28336	9288	13s	17664	28
cnet5	16280	6240	8s	14780	33.4
cnet6	11492	3780	5s	13156	37.6
cnet7	9320	3568	5s	12300	40.2
cnet8	46532	18992	25s	28780	17.2
cnet9	6032	3168	3s	12588	39.3
cnet10	30584	12992	16s	27460	18
cnet11	38008	12040	18s	24768	20
cnet12	1860	1000	1s	11048	44.7
cnet13	1292	660	1s	14276	34.6
cnet14	1748	764	1s	14872	33.2
ccitt1	1328	988	1s	9808	52.5
ccitt3	11384	2684	5s	15316	33.6
ccitt5	7368	1949	3s	15112	34
ccitt6	15992	6260	8s	11892	43.3
ccitt7	560	496	0s.5	33820	15.2

6.2. COMMENTAIRES.

Le codage des escaliers apporte un gain intéressant sur les grands contours, et il devient particulièrement efficace sur les images comportant des cadres ou traits comme cnet2.

Par rapport à la méthode de Johnsen, Segen et Cash (voir chapitre 1, Bell labs, [JOHN]) notre taux de compression sur les images ccitt est meilleur pour ccitt3, 5

et 6 (respectivement 26 30.2 et 41.3 dans la méthode Bell labs), moins bon pour ccitt1 (63.1) et égal pour ccitt7 (15.3).

7. *AFFICHAGE.*

L'affichage par balayage s'est avéré très long en temps d'exécution: ces temps sont de l'ordre de ceux d'extraction des composantes connexes, et sont dus à la mise en place des files d'attente des contours intérieurs et des parois.

L'affichage aléatoire est nettement plus rapide (5 à 50s pour toutes les images), et dans la mesure où l'afficheur possède la fonction d'inversion de plages, on peut lui donner la préférence dans les applications de traitement d'images en noir et blanc.

CONCLUSION

1. PERSPECTIVES IMMEDIATES.

Vu les résultats encourageants obtenus avec la méthode décrite dans ce qui précède, nous poursuivrons notre effort sur:

1. un module de traitement des mini-contours pour obtenir un codage optimal de ce type d'objets,

2. des améliorations sur les algorithmes de comparaison de contours (mieux comprendre l'influence des divers paramètres) ainsi que des combinaisons différentes sur le codage des coordonnées et des références à la bibliothèque, afin d'améliorer encore le taux de compression,

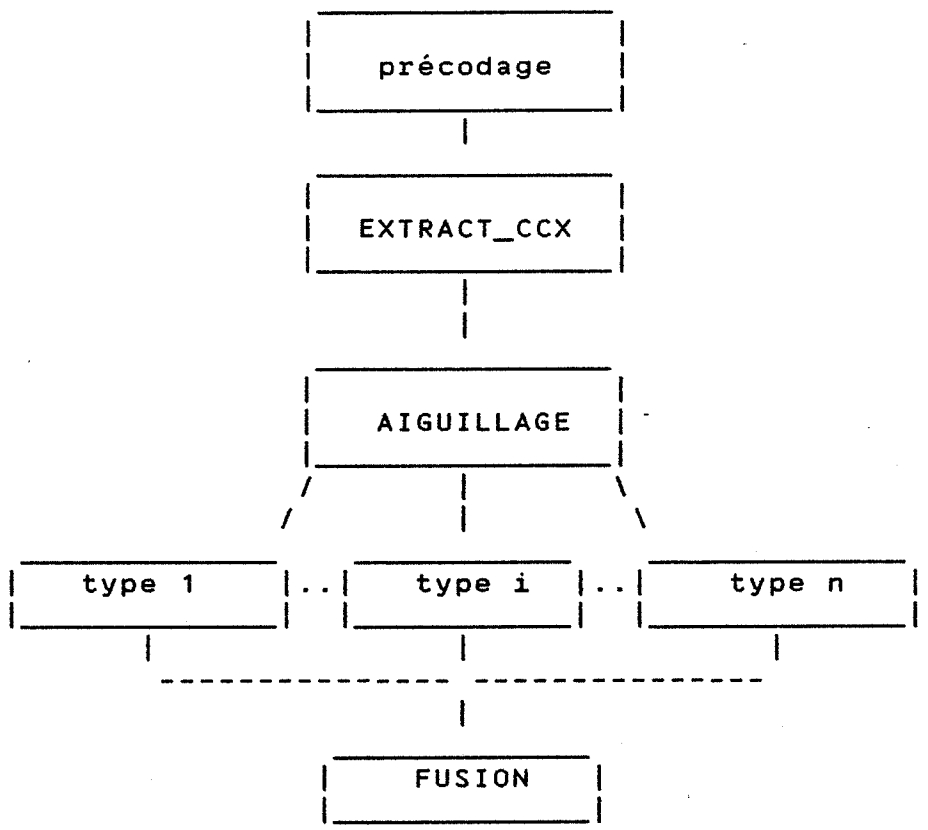
3. une gestion plus adéquate de l'espace dynamique nécessaire lors de l'extraction des grands contours, pour obtenir un gain dans le temps de traitement.

4. une programmation en assembleur du module calculant le précode, avant de passer aux spécifications d'une carte spécialisée ès-précode.

Ceci permettra de définir une machine spécialisée monoprocesseur.

2. A MOYEN TERME.

Avec les réalisations ci-dessus le gain en temps sera surtout obtenu sur la partie précode. Pour le reste il semble intéressant d'utiliser la structure pipe-line des différents modules telle qu'elle est décrite dans le chapitre 4:



2.1. PARALLELISME.

Le traitement en parallèle des différents types de contours peut s'envisager soit en affectant à un processeur donné le processus traitant un type donné de contour, soit en distribuant les processus selon la charge prévue dans le module d'aiguillage. La multiplication du module "petits contours" pose les problèmes suivants:

- soit les différents modules partagent la même bibliothèque, et l'accès à la bibliothèque doit être géré de façon à résoudre les conflits (problème connu de verrouillage lors des mises à jour),

- soit chaque module crée sa propre bibliothèque, et le module de fusion se charge de la récupération des divers morceaux en effectuant une dernière mise à jour.

Remarquons que le module d'extraction des composantes connexes peut lui aussi être dupliqué, et traiter des morceaux d'image avec un découpage arbitraire. La récupération des morceaux de l'image peut se résoudre de la même façon que le traitement d'une suite d'images: Nous proposons un module de synchronisation qui permet une parallèlisation et surtout une certaine vérification du flux d'images traitées.

Un module SYNCHRO communique par messages avec les modules EXTRACT_CCX et FUSION. On peut préciser la nature de ces messages:

EXTRACT_CCX vers SYNCHRO :

3 types de messages:

- j'ai reçu le morceau d'image no i.
- j'ai envoyé le code d'une composante connexe au module d'aiguillage.
- j'en ai fini avec ce morceau d'image.

FUSION vers SYNCHRO :

1 seul type de message:

- j' ai reçu le code d'une composante connexe relatif à l'image no i.

SYNCHRO vers FUSION :

1 seul type de message:

- L'image no i est complète tu peux expédier son code.

AVANTAGES DE CETTE CONCEPTION :

Elle permet d'installer plusieurs modules EXTRACT_CCX en parallèle et assure un premier contrôle du nombre ou de morceaux d'images transitant par le module AIGUILLAGE.

Ce contrôle s'effectue aux deux bouts du traitement d'une composante connexe et le module SYNCHRO avec simple compteur du nombre de composantes connexes associées à une image peut valider le traitement.

2.2. COULEUR.

Le passage à la compression d'images en couleurs peut être envisagé en partant du principe qu'on peut associer à chaque contour une couleur. En remontant la construction des contours, on remarque qu'il faut associer la couleur à chaque paire de parois créée, sans remettre en cause le mécanisme de construction des contours.

Par contre, l'élaboration du précode implique l'étude de n^4-n cas où n est le nombre de couleurs utilisées (qui est déjà réduit du fait que l'on étudie le voisinage 2x2 des interpixels au lieu du voisinage 3x3 des pixels ...).

Faut-il remettre en cause l'affichage aléatoire qui utilise le ou-exclusif et la superposition de contours? Non, si on le traite comme une fonction de retour au fond suivie de l'affichage avec une couleur donnée. Le procédé peut donc être maintenu.

3. CODES DE CONTOURS EN IMAGERIE.

L'utilisation des codes de contours tels que nous les avons définis, par frontière entre objets et extérieur/intérieur d'un objet, paraît très intéressante dans les manipulations d'images. Isoler des formes, avec des traitements séparés de l'intérieur ou du fond, est aisé pusique la frontière n'est pas définie de façon différente pour la forme ou le fond et que l'intérieur ne peut être tangent à l'extérieur. Détecter des éléments singuliers, comme par exemple les escaliers, est nettement simplifié du fait que 4 directions seulement sont définies. La connexité peut être traitée de plusieurs manières sans demander des modfications importantes.

Ainsi la phase d'extraction des composantes connexes en elle-même paraît déjà intéressante en ce domaine. Nous espérons voir bientôt la mise en place de telles applications.

BIBLIOGRAPHIE

[AHRO] AHRONOVITZ E., BERTIER M., HABIB M., Une méthode de compression de documents, Ecole des Mines de St-Etienne, rapport no. 85-9, Avril 1985.

[ASHE] ASHER, NAGY, A means for achieving a high degree of compaction on scan-digitized text. IEEE Trans. Comp., C-23, pp 1174-1179, 1974

[BERG] BERGE C., CHEN C.C, CHVATAL V., SEOW C.S., Combinatorial properties of polyominoes, Combinatorica 1, pp 217-224, 1981

[CED1] CEDERBERG R., Chain link & Segmentation for raster scan devices, CGIP vol 10, 3 (1979), p. 576-578.

[CED2] CEDERBERG R., On the Coding, Processing and Display of Binary Images, Linköping Studies in Science and Technology, Dissertation, No. 57, Linköping, Sweden, 1980.

[DANI] DANIELSSON P., Encoding of binary images by raster-chain-coding of cracks, 6th Int. Conference on Pattern Recognition, Munich 1982, p 335-338.

[EULE] EULER, Solutio Problematis ad Geometriam situs pertinentes, Mémoires Académie des Sciences de Berlin, 1759.

[FREE] FREEMAN H., Computer Processing of Line-Drawing Images. Computing Surveys, Vol. 6, No.1, March 1974.

[GOL1] GOLOMB S.W., Polyominoes, Georges Allen & Unwin ltd, Londres.

[GOL2] GOLOMB S.W., Perfect codes in the Lee metric and the packing of polyominoes, Siam j. of appl. math., vol 18-2,pp 302-317, 1970.

[HUFF] HUFFMAN D. A., A Method for the Construction of Minimum-Redundancy Codes, Proc. of the I.R.E., Vol. 40, p. 1098-1101, 1952.

[JOHN] JOHNSEN O., SEGEN J. and CASH C. L., Coding of Two-Level Pictures by Pattern Matching and Substitution, The Bell System Technical Journal, Vol.62, No.8, pp 2513-2545, 1983.

[JOLY] JOLY P., BRION A., CROCHART K., CHAUVIN G., PICARD M., RAVEL G., Télématisation de la documentation technique, projet d'un système d'archivage et de diffusion, NT/PAA/OGE/TDT/467, CNET-Paris A, 1981.

[KUNT] KUNT M., JOHNSEN O., Codes de Blocs à Préfixe pour la réduction de la Redondance d'Images. Annales des Télécommunications, 33, No. 7-8, 1978.

[MICL] MICLET L. Méthodes structurelles pour la reconnaissance des formes, CNET/ENST, Eyrolles, 1984.

[NADL] NADLER M., Document segmentation and coding techniques, CVGIP 28, pp 240-262, 1984.

[PAVL] PAVLIDIS T., Algorithms for graphics and image processing, Berlin-Heidelberg, Springer-Verlag, 1982.

[PROC] Special Issue on Image Coding. voir en particulier: PRATT et al., YASUDA, PROCEEDINGS OF THE I.E.E.E., Vol. 68, No. 7, July 1980.

[REDE] REDELMEIER D.H., Counting polyominoes: Yet another attack, Discrete Math., Vol 36-2, pp 191-203, 1981.

[ROM1] ROMEO F., JOLY P., Synthèse des expériences de compression, CNET-Paris A, NT/PIA/OGE/TDT/818, note technique, 1983.

[ROM2] ROMEO F., Emploi de Méthodes de Compression et de Reconnaissance dans une Application de Documentation Electronique, Actes du congrès AFCET Reconnaissance des Formes et Intelligence Artificielle, Paris, janvier 1984, tome 2, p 455-463.

[SAIN] A. SAINTE-LAGUE, "Les réseaux (graphes)", Mémorial des Sciences Mathématiques, fasc. 18, Paris, Gauthier-Villars, 1926.

[SEDG] SEDGEWICK R., Quicksort, PH.D, Stanford 1975.

[SIM1] SIMON J.C., La reconnaissance des formes par algorithmes, Paris, Masson, 1984.

[SIM2] SIMON J.C. Invariance en reconnaissance des formes, Cognitiva'85, org. CESTA, Paris, 4-6 Juin 1985.

[TSI] Techniques et sciences informatiques, T.S.I. vol 3, No 1, 1984, Numéro spécial sur le parallélisme.

ANNEXE 1

DETAILS DE LA BIBLIOTHEQUE

Les résultats de la répartition des petits contours dans la bibliothèque sont donnés ci-après. Le contenu des diverses colonnes doit être interprété comme suit:

colonne 1: Nous avons réparti les contours en classes selon la longueur. Cette colonne donne la limite supérieure de chaque classe (la limite inférieure est celle de la classe précédente + 2).

colonne 2: Nombre total de contours dans cette classe.

colonne 3: Nombre de contours n'ayant pas de contours semblables.

colonnes 4 et 5: Eléments de bibliothèque ayant de 1 à 4 semblables chacun. La colonne 4 donne le nombre d'éléments de bibliothèque (i.e nombre de contours différents). La colonne 5 donne le nombre de contours semblables (colonne 4 non comptée).

colonnes suivantes: Comme les colonnes 4 et 5; seul le nombre de contours semblables par élément de bibliothèque est modifié.

Enfin, on rappelle par image le nombre de petits contours global et le nombre de contours constituant la bibliothèque.

cnet1

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	77	8	4	5	1	9	3	48
40	211	20	9	19	4	27	5	127
60	33	13	5	8	1	6	0	0
80	102	40	15	23	1	9	1	13
100	214	93	32	50	1	5	2	31
120	72	32	10	18	0	0	1	11
140	11	11	0	0	0	0	0	0
160	1	1	0	0	0	0	0	0
180	4	4	0	0	0	0	0	0
200	2	2	0	0	0	0	0	0

total contours: 727 élements de bib: 318

cnet2

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	53	7	4	7	3	19	1	13
40	49	32	4	7	1	5	0	0
60	38	32	3	3	0	0	0	0
80	29	29	0	0	0	0	0	0
100	12	12	0	0	0	0	0	0
120	10	5	2	3	0	0	0	0
140	4	2	1	1	0	0	0	0
160	51	30	7	14	0	0	0	0
180	8	6	1	1	0	0	0	0
200	3	3	0	0	0	0	0	0

total contours: 257 élements de bib: 184

cnet3

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	271	17	20	37	2	16	8	172
40	271	100	33	43	10	59	2	24
60	159	117	17	25	0	0	0	0
80	101	69	11	15	1	5	0	0
100	61	57	2	2	0	0	0	0
120	31	31	0	0	0	0	0	0
140	19	19	0	0	0	0	0	0
160	15	15	0	0	0	0	0	0
180	30	30	0	0	0	0	0	0
200	2	2	0	0	0	0	0	0

total contours: 960 élements de bib: 562

cnet4

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	164	22	20	34	7	43	2	37
40	141	72	24	38	1	6	0	0
60	164	119	11	15	1	6	1	11
80	102	82	9	11	0	0	0	0
100	63	52	5	6	0	0	0	0
120	32	28	2	2	0	0	0	0
140	23	23	0	0	0	0	0	0
160	29	27	1	1	0	0	0	0
180	20	20	0	0	0	0	0	0
200	1	1	0	0	0	0	0	0

total contours: 739 élements de bib: 529

cnet5

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	247	22	15	29	4	28	6	144
40	248	104	31	48	5	29	2	29
60	540	90	14	18	1	6	3	408
80	53	42	5	6	0	0	0	0
100	62	53	4	5	0	0	0	0
120	41	39	1	1	0	0	0	0
140	16	16	0	0	0	0	0	0
160	11	9	1	1	0	0	0	0
180	6	6	0	0	0	0	0	0
200	4	4	0	0	0	0	0	0

total contours: 1228 élements de bib: 476

cnet6

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	415	29	21	38	4	26	9	289
40	196	109	24	33	4	26	0	0
60	119	106	6	7	0	0	0	0
80	87	87	0	0	0	0	0	0
100	71	63	4	4	0	0	0	0
120	39	35	2	2	0	0	0	0
140	30	30	0	0	0	0	0	0
160	23	21	1	1	0	0	0	0
180	10	10	0	0	0	0	0	0
200	4	4	0	0	0	0	0	0

total contours: 994 élements de bib: 568

cnet7

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	302	26	16	33	3	23	7	195
40	195	73	25	51	1	5	3	37
60	175	106	12	19	1	5	1	31
80	93	71	10	12	0	0	0	0
100	60	54	3	3	0	0	0	0
120	40	34	3	3	0	0	0	0
140	41	38	1	2	0	0	0	0
160	24	22	1	1	0	0	0	0
180	11	9	1	1	0	0	0	0
200	1	1	0	0	0	0	0	0

total contours: 942 élements de bib: 521

cnet8

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	180	15	16	31	5	29	5	80
40	381	60	14	29	4	26	8	240
60	336	79	21	35	6	40	6	149
80	152	76	22	35	1	5	1	12
100	135	77	12	19	1	5	1	20
120	27	19	3	5	0	0	0	0
140	15	15	0	0	0	0	0	0
160	33	31	1	1	0	0	0	0
180	14	14	0	0	0	0	0	0
200	3	3	0	0	0	0	0	0

total contours: 1276 élements de bib: 515

cnet9

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	299	22	15	34	3	22	7	197
40	216	72	28	44	6	40	2	24
60	142	96	16	23	1	6	0	0
80	93	59	7	11	2	14	0	0
100	110	76	8	20	1	5	0	0
120	103	37	3	8	2	15	1	37
140	94	34	1	1	0	0	3	55
160	17	17	0	0	0	0	0	0
180	8	8	0	0	0	0	0	0
200	2	2	0	0	0	0	0	0

total contours: 1084 élements de bib: 528

cnet10

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	871	14	15	35	10	60	9	729
40	699	71	46	77	6	43	7	449
60	239	119	14	17	5	27	3	54
80	157	96	17	27	2	15	0	0
100	101	74	10	17	0	0	0	0
120	60	42	4	6	1	7	0	0
140	41	35	3	3	0	0	0	0
160	36	36	0	0	0	0	0	0
180	23	23	0	0	0	0	0	0
200	8	8	0	0	0	0	0	0

total contours: 2235 élements de bib: 669

cnet11

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	508	10	22	44	8	56	5	364
40	542	61	27	45	2	12	8	387
60	214	106	24	37	4	23	1	19
80	129	75	11	13	2	14	1	13
100	57	46	5	6	0	0	0	0
120	27	24	1	2	0	0	0	0
140	31	25	2	4	0	0	0	0
160	202	118	9	11	1	5	3	55
180	12	10	1	1	0	0	0	0
200	2	2	0	0	0	0	0	0

total contours: 1724 élements de bib: 613

cnet12

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	303	10	8	13	4	27	5	237
40	279	19	7	12	3	20	6	212
60	112	14	3	5	3	18	2	67
80	498	79	40	69	5	30	9	266
100	381	97	29	48	7	45	7	148
120	16	12	2	2	0	0	0	0
140	8	8	0	0	0	0	0	0
160	12	12	0	0	0	0	0	0
180	10	10	0	0	0	0	0	0

total contours: 1619 élements de bib: 400

cnet13

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	497	16	5	11	4	25	7	430
40	425	28	16	27	3	23	6	322
60	168	30	6	9	2	12	3	106
80	762	95	56	106	8	54	10	433
100	515	114	39	77	4	26	8	247
120	22	22	0	0	0	0	0	0
140	5	5	0	0	0	0	0	0
160	12	12	0	0	0	0	0	0
180	11	11	0	0	0	0	0	0

total contours: 2417 élements de bib: 509

cnet14

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	458	8	7	11	4	26	5	398
40	427	29	7	13	6	40	7	325
60	129	21	5	8	0	0	3	92
80	852	104	49	93	9	59	13	525
100	479	120	48	83	8	47	7	166
120	29	25	2	2	0	0	0	0
140	11	11	0	0	0	0	0	0
160	12	12	0	0	0	0	0	0
180	11	11	0	0	0	0	0	0

total contours: 2408 élements de bib: 520

ccitt1

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	243	9	5	10	2	13	6	199
40	221	32	11	21	2	15	5	135
60	329	87	28	43	8	59	3	101
80	396	107	34	58	6	39	8	144
100	119	40	5	6	3	19	2	44
120	23	17	3	3	0	0	0	0
140	5	5	0	0	0	0	0	0
160	4	4	0	0	0	0	0	0
180	3	3	0	0	0	0	0	0
200	1	1	0	0	0	0	0	0

total contours: 1344 élements de bib: 435

ccitt3

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	243	10	9	19	0	0	6	200
40	370	39	17	30	6	44	9	225
60	460	120	53	98	8	54	6	121
80	386	119	29	47	6	45	5	135
100	206	83	27	49	3	21	2	21
120	61	48	5	8	0	0	0	0
140	8	8	0	0	0	0	0	0
160	5	5	0	0	0	0	0	0
180	3	3	0	0	0	0	0	0

total contours: 1742 élements de bib: 625

ccitt5

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	406	11	9	18	5	34	8	322
40	597	58	29	51	7	47	13	392
60	738	126	47	77	9	63	11	405
80	521	135	50	95	8	50	8	175
100	78	40	7	9	1	6	1	14
120	35	25	2	2	1	5	0	0
140	1	1	0	0	0	0	0	0
200	3	3	0	0	0	0	0	0

total contours: 2379 élements de bib: 614

ccitt6

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	119	14	7	10	5	26	4	54
40	175	69	24	40	4	25	1	12
60	182	103	25	40	2	12	0	0
80	73	47	10	16	0	0	0	0
100	40	28	6	6	0	0	0	0
120	14	14	0	0	0	0	0	0
140	8	6	1	1	0	0	0	0
160	3	3	0	0	0	0	0	0
180	6	6	0	0	0	0	0	0
200	1	1	0	0	0	0	0	0

total contours: 621 élements de bib: 379

ccitt7

classe de lg	total classe	sans sembl	de 1 à 4 bib	de 1 à 4 sembl	de 5 à 9 bib	de 5 à 9 sembl	+ de 10 bib	+ de 10 sembl
20	1598	21	22	48	8	56	16	1428
40	1474	161	110	211	20	134	26	812
60	905	421	98	147	15	105	3	116
80	519	318	48	78	4	25	3	43
100	352	202	25	36	2	17	3	67
120	124	66	6	7	1	5	2	37
140	55	55	0	0	0	0	0	0
160	23	23	0	0	0	0	0	0
180	3	3	0	0	0	0	0	0
200	1	1	0	0	0	0	0	0

total contours: 5054 élements de bib: 1682

ANNEXE 2

IMAGES CNET

Nous présentons les images "origine", fournies par le CNET, avec lesquelles nous avons effectué les essais. Elles ont été numérisées avec un facteur de résolution de 8 points par mm. Ne pas tenir compte de l'intitulé "cnetx" en tête d'image.

3.3.5 Tarif nominal : périodes nominales des sorties

3.3.6 Valeur des coefficients diviseurs de la période

3.3.7 Affectation des coefficients aux sorties

3.3.8. Passage en phase active.

3.4 Phase Active

3.4.1 Commande principale

3.4.1.1 Impression des paramètres IMP

3.4.1.2 Liste et modification des paramètres LMP

3.4.1.3 Visualisation de paramètre VIS

3.4.1.4 Demande de commande spéciale TST

3.4.2 Commandes secondaires

3.4.2.1 Tous les paramètres de taxation TOU

3.4.2.2 Type de Central Téléphonique TYP

a) IMP
b) LMP
c) VIS

3.4.2.3 Code du central CEN

a) IMP
b) LMP
c) VIS

3.4.2.4 Date DAT

a) LMP
b) VIS

3.4.2.5 Heure HEU

a) LMP
b) VIS

D C B A

4

MF/éd

ORIGINAL A4. 10055 MA édit

1

MXX

des. :	éditions	1		4		7		10					
ver. :		2		5		8							
appr. :		3		6		9							

gestion

cnet CENTRE PARIS A

page /

D C B A

REPÈRE	DÉSIGNATION	RÉFÉRENCE		QUANTITÉ	OBSERVATIONS
1	STRUCTURE MODULAIRE SDT	MXX10042.01		1	
	Pupitre Dialog 80 Enfichable				
	Tronc commun.				
2	Tiroir avec imprimante.	M10047-02		1	
3	Carte interface 48V.	M10049-02		1	
4	Câble prolongateur pour boîtier	M13419-00		1	
	DIL. 24 broches (Mâle-Mâle).				
5	Etiquette (positionnement)	MAV12414		1	
6	Etiquette d'individualisation	MAY12723		1	
7	Circuit intégré programmé	INTEL	B 2708	1 *	
	Etiquette suivant dossier	MXX30000-00-02		1	
01.05	Capot pour bouchon (M10043)	COMATEL	30-16-200	3	
0206					
0307	Etiquette auto-collante 12x12	MAY12722		3	
	Cliché de fabrication	M10256A.01.A			

* Contenu Cassette - M 30 000-02
MEMOIRE REPROM
Génératrice de caractères

MXX10256-A-01

Des CGCT	Editions	1	82.10.13
Ver		2	83.02.15
Appr			

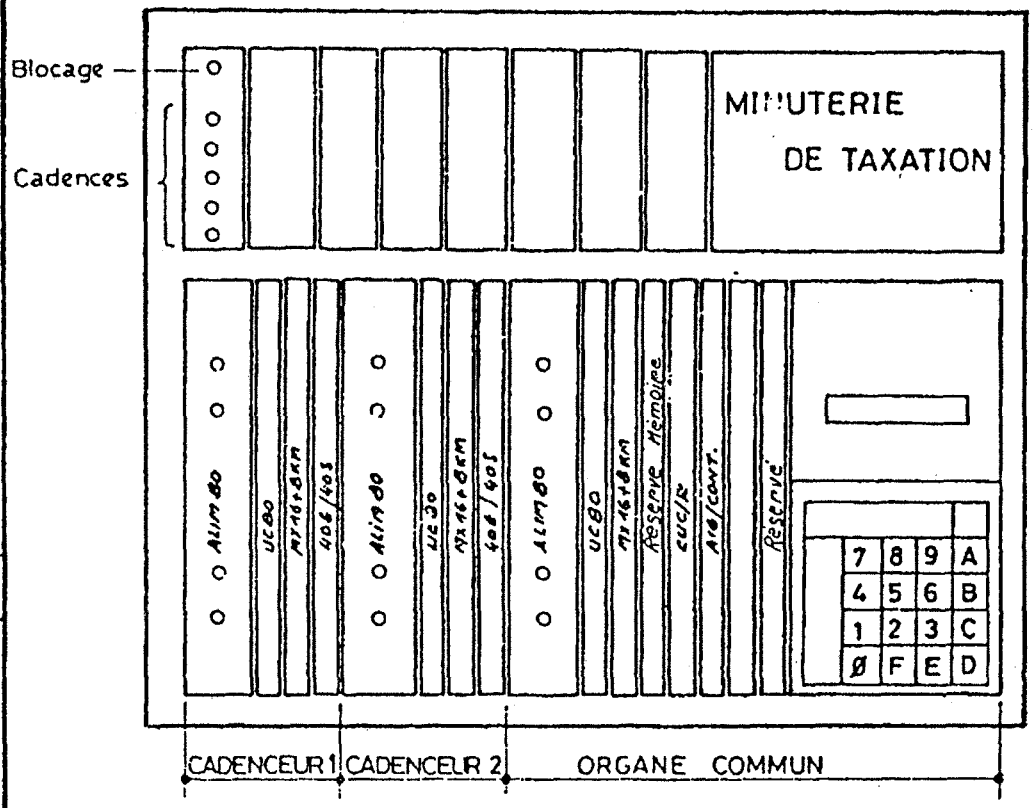
Utilisation

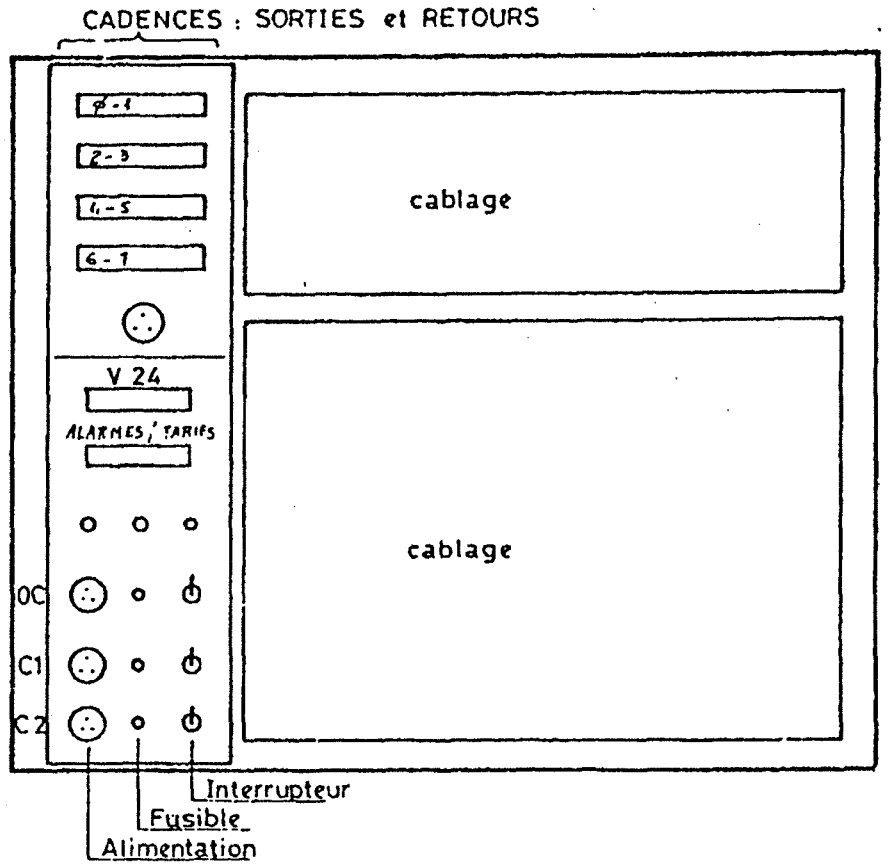
Gest: CNET

.NOMENCLATURE du PLAN MRM10471

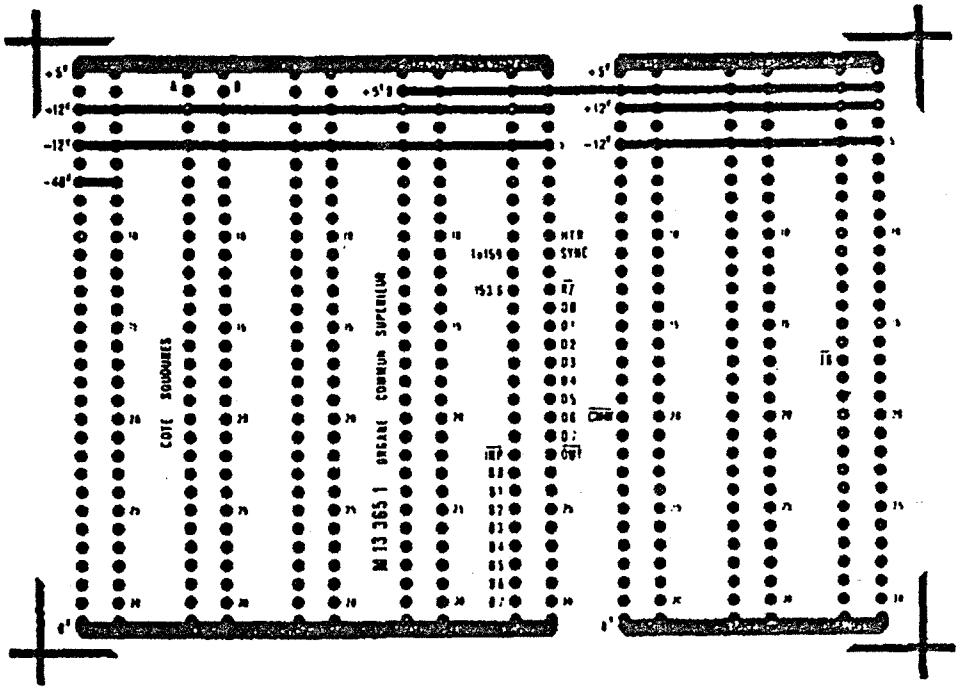
Page: MAM12413

COFFRET DE LA MINUTERIE DE TAXATION





Editions CNET MAN 12 058



496 x A

TROUS METALLISES X		TROUS NON METALLISES		CLASSE D'IMPRESSION 2
Traitement	Connecteur	Ni. Au		Ref. Decoupe aux traits
	Carte	Sn. Pb X	Electrolytique X	+ Surfusion
MATIERE	Support stratifié	Ep 1,6 mm	Ref EPGC2	Cu 2 x 35 µ
Percage	8	A: 0 8 / 1 / 1 3 / 1 5 / 2 2 / 2 7		
Dessinateur BARRAS. P	Modifications	1 82.03.31		

MXX 13 292.

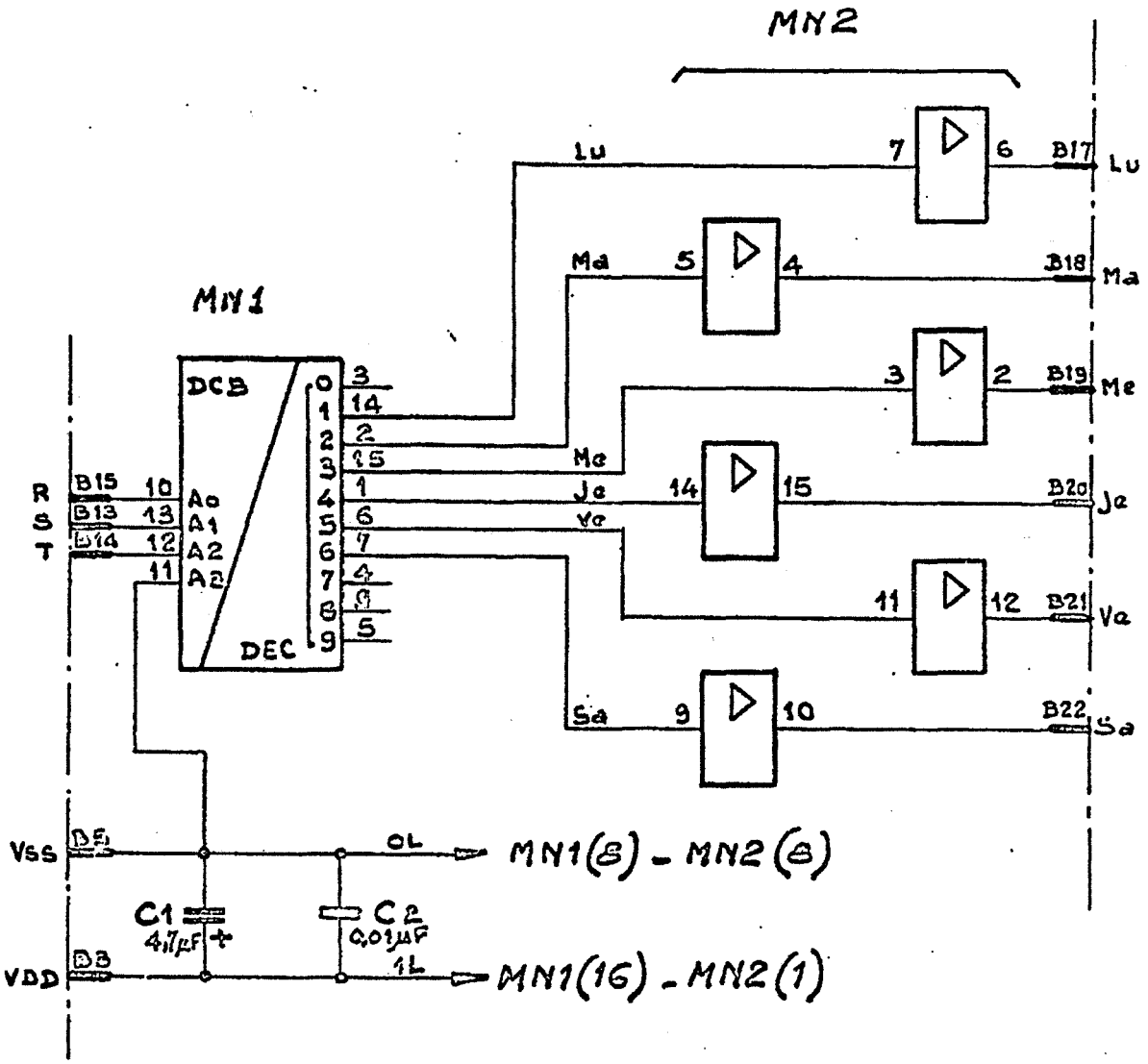
BARRAS PROVENCE

PERCAGE CARTE O.C. SUPERIEUR

CNET

MAU 12 259

OR. 257 MA



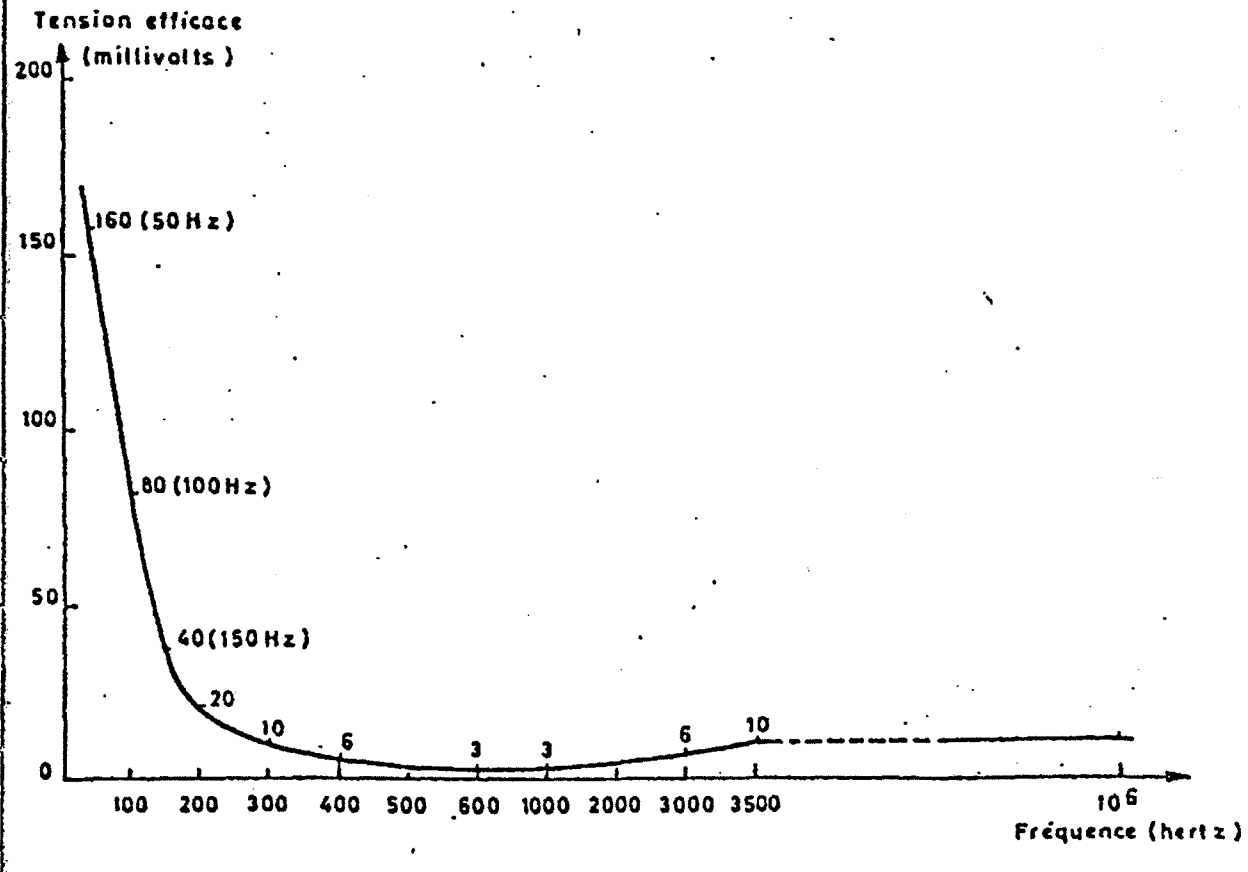
Des. Pauly	Editions	1	4	7	10	13	16
Ver. Rouyer		2	5	8	11	14	17
Appr.		3	6	9	12	15	A : 79.02.01

CARTE DE DECODAGE
SCHEMA DE PRINCIPE

Gest. CNET RC

Page MAN10.1

OR. 257 MA



TENSIONS ALTERNATIVES ADMISSIBLES SUR L'ALIMENTATION 48 V DES ÉQUIPEMENTS TÉLÉPHONIQUES

Copie du document CNET. KAX.9516

Des Liandier	Editions	1. 80.07.30	4	7 :	10 :	13 :	16
Ver Roupé		2	5	8 :	11 :	14 :	17
Appr.		3	6	9	12 :	15 :	

GENERATEUR DE TONALITE
(330 + 440) Hz
. COURBE .

Gest. CNET

Page MAX 10254

UTILISATION MXX 13 294 A01 — MF/Ed

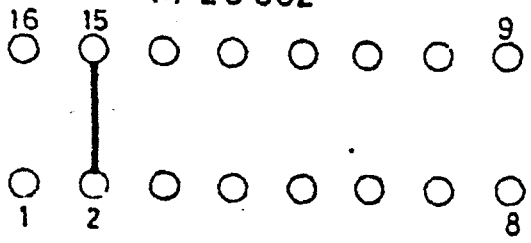
Editions 1 2 — Page 4

	Fonction	FILS Couleur	FILS Nature	Points de connexion	Observations
1	⏚	N	KY	(B) 117/002 - (B) 217/002 - (B) 218/002 - (B) 118/002 - (B) 119/002 -	
2			33A03	- (B) 219/002 - (B) 131/010.	
3				(B) 321/002 - (B) 421/002 - (B) 422/002 - (B) 322/002 - (B) 323/002 -	
4				- (B) 423/002 - (B) 231/010.	CADENCEUR 1
5				(B) 219/002 - (B) 331/010.	
6				(B) 431/010 - (B) 427/002 - (B) 415/002.	
7					
8	⏚	N	KY	(B) 117/019 - (B) 217/019 - (B) 218/019 - (B) 118/019 - (B) 119/002 -	
9			33A03	- (B) 219/019 - (B) 131/027.	
10				(B) 321/019 - (B) 421/019 - (B) 422/019 - (B) 322/019 - (B) 323/019 -	
11				- (B) 423/019 - (B) 231/027.	CADENCEUR 2
12				(B) 219/019 - (B) 331/027.	
13				(B) 431/027 - (B) 427/019 - (B) 415/019.	
14					
15	⏚	N	KY	(B) 117/036 - (B) 217/036 - (B) 218/036 - (B) 118/036 - (B) 119/036 -	
16			33A03	- (B) 219/036 - (B) 131/044.	
17				(B) 421/036 - (B) 321/036 - (B) 322/036 - (B) 422/036 - (B) 423/036 -	
18				- (B) 323/036 - (B) 231/044.	
19				(B) 219/036 - (B) 331/044.	
20				(B) 431/044 - (B) 427/036 - (B) 415/036.	ORGANE COMMUN
21				(B) 118/036 - (B) 131/053.	
22				(B) 421/036 - (B) 231/053.	
23				(B) 119/036 - (B) 331/053 - (B) 431/053 - (B) 331/056 - (B) 431/056 -	
24				- (B) 331/059 - (B) 431/059 - (B) 331/065 - (B) 431/065.	
25					
26					
27	+ 12 V	BL	KY	(B) 324/002 - (B) 424/002 - (B) 425/002 - (B) 325/002 - (B) 326/002 -	CADENCEUR 1
28		1	33A03	- (B) 426/002 - (B) 103/010.	
29					
30					

CNET MAY 1117411

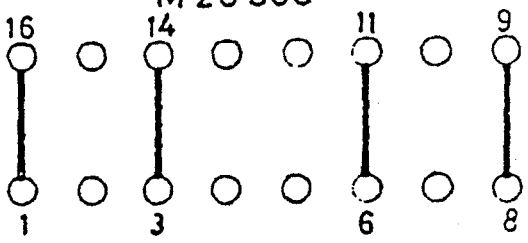
cnet8

M 20002



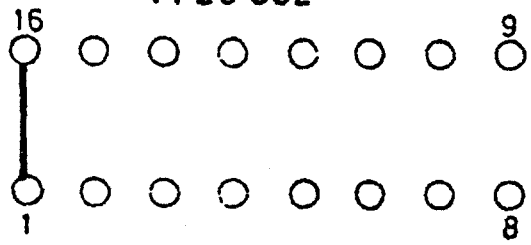
POSITION 01-05

M 20008

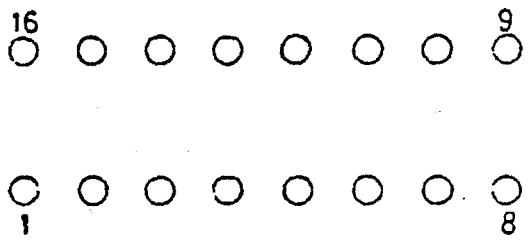


POSITION 02-06

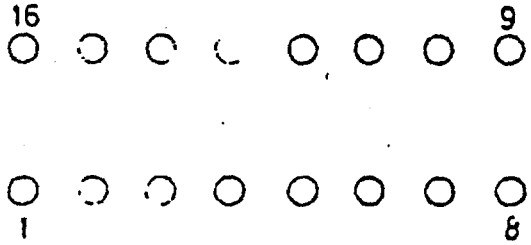
M 20052



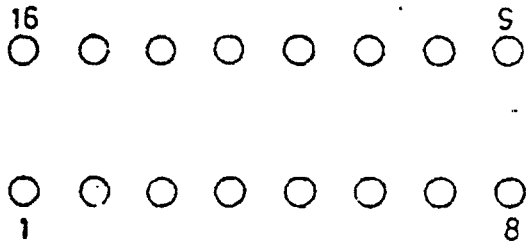
POSITION 03-07



POSITION



POSITION



POSITION

Carte : Logique M 10043

MXX10256-A-01	Des:BARRAS.	Edition	1 82.03.31	4	7	10	13	16	BARRAS PROVENCE		Gest	CNET				
MXX 10 219	Ver.		2 83-02-15	5	8	11	14	17								
UTILISATION	Appr		3	6	9	12	15	18	PUPITRE DIALOG-80		Page	MAY	11 955			
									Cablage des bouchons							

OR.10002 MA

N° PLAN	DESIGNATION	Dp	EDITION DU DOSSIER 1 2
MXX 10256-A-01	Page de garde.		1 2
MXX 10256-A-01	Éditions successives.		1 2
MXX 10256-A-01	Liste des documents.		1 2
MAM 12413	Nomenclature.		1 2
MRM 10471	Équipement.		1 1
MAV 12414	Positionnement du marquage		1 1
MAY 11955	Cablage bouchon MXX10043		2
MAY 12722	Etiquette bouchon		1
MAY 12723	Etiquette d'individualisation		1
MZZ 10256-A	Fiche d'évolution de produit.	X	

INDICE "LETTRE" DE LA CLASSE D'EXPLOITATION	A A

codification	désignation	dp	éditions du dossier 1	2	3	4	5	6	7	8	9	10
MXX 13411	Page de garde 1/3			2	3							
MXX 13411	Editions successives 2			2	3							
MXX 13411	Liste des documents 3/3			2	3							
MAN 12313	Notice relative aux cartes			2	3							
MAN 12314	Notice relative au cablage			2	2							
MAN 12315	Notice relative aux composants			2	2							
MAS 12355	Etiquette de marquage			1	1							
CGV 1135	Pontet			7	7							
LAR 0274	Fil 4/10 (3p)			6	6							
LAR 0414	Fil 8/10 (6p)			1P	1P							
LAR 0430	Fil paires 4/10 (3p)			1b	1b							
CGB 6924	Cable souple isolé (4p)			11	11							
LAR 0604	Etiquette			2	2							

classe d'exploitation

LISTE DES DOCUMENTS | page 3/3 | cnet CENTRE PARIS A | MXX 13411

MF/éd

MXX

ORIGINAL A4 10002 MA 841

SOMMAIRE

Le processus de la reconnaissance d'une forme est un processus de prise de décisions en un certain nombre d'étapes. Les opérateurs et formules logiques permettent la caractérisation d'un ensemble à l'aide d'un petit nombre de variables. Ceci simplifie et rend plus facile la compréhension des problèmes de décision. La logique floue introduit la subjectivité dans l'exécution des programmes informatiques.

Etant donné un ensemble de classes d'objets décrits avec des formules logiques nous présentons ici un algorithme qui permet d'obtenir une caractérisation des classes et, à partir de celle-ci, un ensemble d'opérateurs de reconnaissance. Il est donné un exemple illustratif.

APPLIED LOGIC IN PATTERN RECOGNITION

Abstract

The process of recognizing a pattern is a process of decisions taken in a number of steps. Logic operators and formulae allow the characterization of a set by means of a small number of variables. This simplifies and makes it easier to understand decision making problems. Fuzzy logic introduces subjectivity in the execution of computer programs.

The algorithm presented here shows how, given a set of classes of objects described with logic formulae, one can achieve, firstly, a characterization of classes and, secondly, a set of recognition operators. An illustrative example is given.

1. INTRODUCTION

Les fondements des mathématiques sont le problème principal de la logique moderne. Mais la logique s'occupe aussi d'autres problèmes comme la description de raisonnements et phénomènes. La logique appliquée a peut-être moins de prestige que la logique pure ; néanmoins l'informatique, entre autres sciences, connait grâce à elle un nouveau développement et de nouvelles perspectives.

En Reconnaissance de Formes (R.de F.) un algorithme qui traite les données initiales pour les rendre plus facilement interprétables est une application d'un espace de Représentation dans un espace d'Interprétation.

En R. de F. nous avons fréquemment le problème de l'affectation des objets qui ne font pas partie de l'ensemble d'apprentissage. Nous proposons ici une interprétation d'une partition de cet ensemble avec des formules logiques, laquelle permet d'obtenir des opérateurs de reconnaissance.

2. UN PEU D'HISTOIRE

Avec les préoccupations philosophiques et mathématiques de son époque, Aristote (384-322 av. J.C.) prit conscience de la structure de la logique des classes et définit les notions de vrai et de faux à côté desquelles l'incertitude ne pouvait pas exister. Après plusieurs siècles, vers 1847, la notation de Boole permit de manipuler, à la manière d'entités mathématiques représentées par des symboles, les propositions logiques exprimées dans le contexte général de la logique d'Aristote.

Vers 1870 Cantor exposa la théorie des ensembles. Plus tard, l'algèbre de Boole a été érigée en outil de travail au service des fonctions univoques à deux états possibles selon la technique de Shannon. Les connaissances léguées par Pascal, Hollerith, Babbage, Leibnitz, ..., ont permis à Aiken la construction d'un prototype d'ordinateur vers 1938, perfectionné ensuite par lui-même et tant d'autres, avec l'impact que nous connaissons aujourd'hui.

.../...

Cette fonction d'appartenance avec des valeurs dans l'intervalle [0,1] ouvre le chemin à une généralisation du concept d'ensemble, applicable dans le domaine de la R. de F., lorsqu'on évalue le risque de prendre une décision. La programmation et l'apprentissage s'enrichissent avec les concepts flous.

Le processus de la reconnaissance d'une forme s'identifie à une prise de décisions en un certain nombre d'étapes. En classification automatique et R. de F., les notions de distance, similarité, niveau de perception ,..., sont largement employés . Avec l'application d'un schéma logique, l'ordinateur est doté de moyens qui lui permettent d'acquérir et de développer certains talents cognitifs ; il peut faire un apprentissage et l'améliorer, il peut manipuler des expressions et analyser des structures logiques.

L'être humain apprend à la machine, avec un programme, ce qu'elle doit connaître, en lui indiquant ce qui à ses yeux semble être des groupes d'objets. L'ordinateur calcule le nombre de groupes, des distances, associe les différents centres d'un échantillonnage à l'autre, détermine la similarité entre objets, bref, classifie.

La méthode des Nuées Dynamiques de DIDAY [1] (en anglais Dynamic Clusters Method) s'emploie avec succès pour de nombreux types de problèmes en classification automatique.

3. OBSERVATION ET RECONNAISSANCE DE FORMES

La R. de F. apparaît de plus en plus comme l'art de relier un signifié à un signifiant.

Une *observation* est une proposition qui peut être formulée sur des objets d'un ensemble. L'observation est vérifiée par les objets et est le support de cette qualité ; un objet peut vérifier un peu, beaucoup, plus ou moins, ou pas du tout l'observation.

Watanabé et Zadeh, entre autres, ont abordé le problème de l'expression d'observations en les représentant avec des fonctions à valeur réelle, Zadeh dans l'intervalle [0,1] et Watanabé en utilisant des fonctions caractéristiques généralisées.

Année 1985 N° d'ordre 173

Université de Saint-Etienne Ecole des Mines de Saint-Etienne

ALGORITHMES DE COMPRESSION D'IMAGES ET CODES DE CONTOURS

par Ehoud AHRONOVITZ

RESUME

Le volume important de la représentation numérique des images pose (entre autres) le problème de leur codage sous forme condensée. Nous avons développé une méthode permettant de construire un code fortement comprimé à partir d'une image bicolore. Elle permet d'extraire tous les objets de l'image en un seul balayage tout en attachant à chaque objet des caractéristiques de sa forme. Des traitements de compression différents sont alors adaptés à chaque type d'objet. Grace à une technique d'affichage reposant sur des principes simples, des opérations algébriques sont possibles sur les contours extraits. L'adéquation à tout format d'image et à toute précision de la saisie est inhérente au principe même de la méthode. Nous proposons un découpage en modules indépendants, en ayant pour objectif une parallélisation ultérieure. Le logiciel, en version monoprocesseur, a été réalisé sur une SM90 (CNET), sous le système Unix™, et donne des résultats souvent meilleurs ou comparables à ceux obtenus par les méthodes connues actuellement.

Mots-clés:

Affichage
Balayage
Composante connexe
Compression
Contour
Image numérique
Télécopie